Machine Vision and Applications (2023) 34:8
https://doi.org/10.1007/s00138-022-01356-0

ORIGINAL PAPER

Real-time pedestrian pose estimation, tracking and localization for social distancing

Bilal Abdulrahman[1] · Zhigang Zhu[2]

Received: 7 July 2021 / Revised: 28 May 2022 / Accepted: 3 November 2022 / Published online: 5 December 2022

Abstract
The corona virus pandemic has introduced limitations which were previously not a cause for concern. Chief among them are wearing face masks in public and constraints on the physical distance between people as an effective measure to reduce the virus spread. Visual surveillance systems, which are common in urban environments and initially commissioned for security surveillance, can be re-purposed to help limit the spread of COVID-19 and prevent future pandemics. In this work, we propose a novel integration technique for real-time pose estimation and multiple human tracking in a pedestrian setting, primarily for social distancing, using CCTV camera footage. Our technique promises a sizeable increase in processing speed and improved detection in very low-resolution scenarios. Using existing surveillance systems, pedestrian pose estimation, tracking and localization for social distancing (PETL4SD) is proposed for measuring social distancing, which combines the output of multiple neural networks aided with fundamental 2D/3D vision techniques. We leverage state-of-the-art object and pose estimation algorithms, combining their strengths, for increase in speed and improvement in detections. These detections are then tracked using a bespoke version of the FASTMOT algorithm. Temporal and analogous estimation techniques are used to deal with occlusions when estimating posture. Projective geometry along with the aforementioned posture tracking is then used to localize the pedestrians. Inter-personal distances are calculated and locally inspected to detect possible violations of the social distancing rules. Furthermore, a "smart violations detector" is employed which estimates if people are together based on their current actions and eliminates false social distancing violations within groups. Finally, distances are intuitively visualized with the right perspective. All implementation is in real time and is performed on Python. Experimental results are provided to validate our proposed method quantitatively and qualitatively on public domain datasets using only a single CCTV camera feed as input. Our results show our technique to outperform the baseline in speed and accuracy in low-resolution scenarios. The code of this work will be made publicly available on GitHub at https://github.com/bilalze/PETL4SD.

Keywords Multiple object tracking · Human pose estimation · Pedestrian localization

1 Introduction

It has been over two years since the initial cases of the COVID-19 virus surfaced [33]. The world is now still amidst a global pandemic. Even though multiple vaccines have been introduced around the world and are being swiftly implemented, new variants which are deadlier and spread faster keep appearing [11]. The pandemic has introduced limitations which were previously not a cause for concern. Chief among them are wearing face masks in public and social distancing measures. In public health, social distancing is a set of non-pharmaceutical interventions or measures intended to prevent the spread of a contagious disease by maintaining a physical distance between people and reducing the number of times people come into close contact [17]. The CDC (Center for Disease Control and Prevention) in the USA recommends 6 feet as a safe distance between individuals. Visual surveillance systems, which are common in urban environments, aim at providing safety in everyday life. High-quality surveillance cameras are already present in most urban streets and departmental stores. Although they were initially commis-

✉ Bilal Abdulrahman
babdulrahman@gradcenter.cuny.edu

Zhigang Zhu
zzhu@ccny.cuny.edu

1 The Graduate Center, The City University of New York, New York, NY 10016, USA

2 The City College and The Graduate Center, The City University of New York, New York, NY 10031, USA

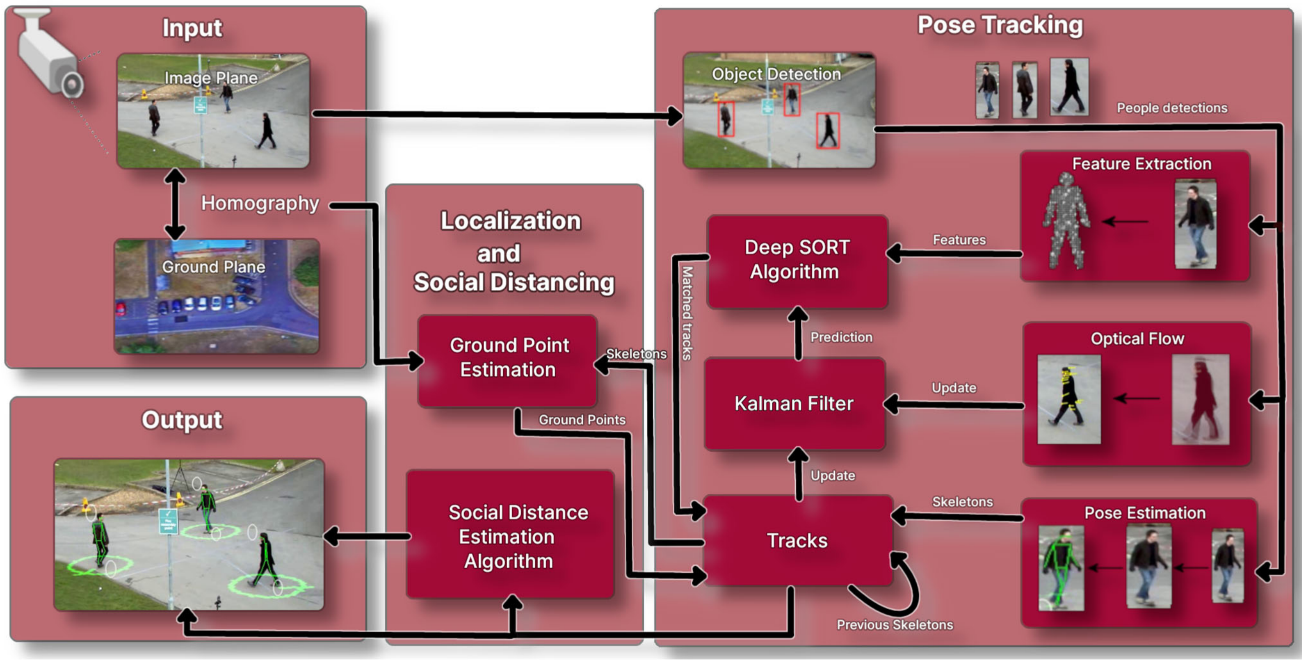

Fig. 1 Workflow of the PETL4SD system pipeline: pedestrian pose estimation, tracking and localization for social distancing

sioned for security surveillance, they can be re-purposed to help limit the spread of COVID-19 and prevent future pandemics (Fig. 1).

In this work we propose a novel and systematic approach to social distancing measurement in a pedestrian setting, called PETL4SD, which stands for Pose Estimation, Tracking and Localization for Social Distancing. This technique functions as it states in its name. Pose estimates are tracked, which are then used for localization, and thus, inter-personal distances are calculated and social distancing violations are estimated. That being said, this is a high-level overview and there are many nuances that add to the novelty of our technique. These will be explained in the coming sections. Furthermore, although we employ PETL4SD for social distancing estimation in this work, its modular nature allows it to be generalized for a bevy of applications, such as detecting and tracking human behavior, action estimation, analyzing pattern of movement (for example in clothing shops), training pose estimator with labels, etc.

During initialization of the PETL4SD pipeline, we compute homography or a projective transformation between the ground plane in the real world and ground plane as it is represented in the image plane of the camera. Homography is required to be computed only once and can be done before hand. We will use the estimated homograph along with the point of intersection of pedestrians with the ground plane (i.e., their feet) to localize them. After initial setup, our technique utilizes a pipeline of tasks that are performed on each frame of the video feed. These tasks are as follows: (1) pedestrian detection; (2) pedestrian pose estimation; (3) pedestrian feature extraction; (4) pedestrian track/tracks creation/update; (5) pedestrian localization; (6) social distancing calculation; and (7) visualization of the tracks and social distance violations.

In the first step, a neural network object detector, in this case YOLOv5 [19], is used for the detection of pedestrians. In the second step, the detected bounding boxes of pedestrians are resized and then fed to the OpenPifPaf network [20] for posture estimation. The detections from the object detector are also used to extract features in step three using the OSNET REID network [37] for use in tracking and re-identification. All models used in this paper are pretrained models; for training parameters and training methodology please refer to the respective publications. Tracking process begins in the fourth step, and the tracking method used is similar to FASTMOT [36] with modifications made for this implementation. This technique uses a custom version of the DeepSORT algorithm [34] complimented with optical flow estimation. In the fifth step, we update the tracks or create new ones by matching the detected pedestrian bounding boxes, the optical flow points and the REID template with the previous frame. Then we employ our temporal and analogous techniques for pose completion to estimate feet of pedestrians if not detected due to occlusion. Homography is then used to project the feet coordinates from the image onto real-world coordinates and, thus, localize the pedestrians. In the sixth step, the ground points history of the pedestrians is used to estimate whether there has been a social distancing violation. Finally, the seventh step is when the distances between pedestrians are visualized with the right perspective and occlusion relations using a simple Augmented Reality (AR) composition of the real scene background, the social distance marks on the ground and the images of the detected pedestrians, as if the marks are actually on the ground.

Along with other improvements, the main contribution of our work is:

(1) We exploit the standard surveillance setting, where people appear up-right and this heuristic allows us to implement a bottom-up approach to pose estimation as essentially a top-down approach. Using prepossessing tricks we are able to get a sizeable increase in speed and improvement in pose estimation, making the system function in real time on less capable hardware.
(2) We utilize posture tracking history along with localized mean distributions of human body dimensions to estimate the position of feet even if they are not detected due to occlusions. Even if the pose detector is unable to detect any posture, the posture and bounding box history are used to get a reasonable estimate of the pedestrians feet position and thus improve localization accuracy.
(3) We created a "smart violations detector" which uses the track and posture history to infer the pedestrians current action and uses this knowledge to group people as "together." This is vital for eliminating false social distance violations between pedestrians in a group.

Furthermore, this approach only requires 4 points of correspondence on the ground visible in the video feed for estimating the planar homography, which can easily be sourced from online maps (if the view is outdoors) or floor plans (if the view is indoors). If correspondence cannot be established, a camera model with intrinsic and extrinsic parameters will also suffice. Therefore, after the initial setup, this approach will work requiring only the video feed. All tests and implementations are done on Python. All models are run using the PyTorch library [24], and most of the computer vision tasks are performed using the OpenCV library [8].

The paper is organized as follows: Sect. 2 discusses some related work. Section 3 describes the mapping from 2D images to the ground plane for a foundation of social distancing measures in the metric space. Section 4 discusses our choice of deep learning model for real-time human bounding box detection. Section 5 presents our integrated solution for human pose estimation with the YOLO object detector and the bottom-up pose estimator—OpenPifPaf. Section 6 describes the combination of DeepSORT with optical flow estimation for human pose tracking. Section 7 presents our temporal and analogous techniques for pedestrian localization and distance calculation. Sections 8 and 9 describe two final steps—social distancing measurement and visualization. Some implementation details and experimental results (including a video demo) are presented in Sect. 10. Finally Sect. 11 concludes the work with discussions of some future directions.

2 Related work

Due to the importance of social distancing measures, a number of works have appeared since the beginning of the pandemic. These works tackle problems such as human detection, human tracking, human localization and combination of such techniques for social distancing monitoring. One such work [28] focuses on detecting small and close by humans from CCTV footage for the purpose of social distancing monitoring. They propose a SSD architecture for this task. Although presenting novel techniques, the work does not highlight its performance when compared to existing techniques and no posture estimation is performed for more accurate tracking and localization estimation is trivial. Another work [2] utilizes transfer learning on YOLOv3 pretrained models to better detect human bounding boxes and then uses tracking to keep track of the violations. This work again lacks localization accuracy and only bounding boxes are tracked leading to vague estimations of the humans actual feet position. Another technique [3] proposes a wearable smart tag to detect and alert for social distancing violations. This is helpful for health authorities to implant on a known COVID patient. However, it can be cause for a privacy concern and large-scale adoption does not seem feasible. A social distancing monitoring approach [27] that utilizes YOLOv3 and DeepSORT on CCTV footage was proposed to detect and track pedestrians followed by calculating a violation index for non-social distancing behaviors. The way this approach estimates the position of pedestrians is unclear as there is no implementation discussion on this part and the work mostly focuses on the DeepSORT algorithm [34].

Several prototypes utilizing machine learning and AI along with cameras and sensors have been developed as commercial solutions for social distancing monitoring and enforcing. Landing AI [21] has proposed a social distancing detector using a surveillance camera to highlight social distancing violations. The work in [26] details a systems that was deployed in a manufacturing plant to monitor worker activity. This type of implementation requires knowledge of a lot more camera parameters and complex transformations compared to our work.

In addition to surveillance cameras, LiDAR-based [16] and stereo camera-based [31] systems are also commercially deployed. These approaches require special hardware made for the specific applications. A recent work [35] proposes a non-intrusive warning system with softer omnidirectional

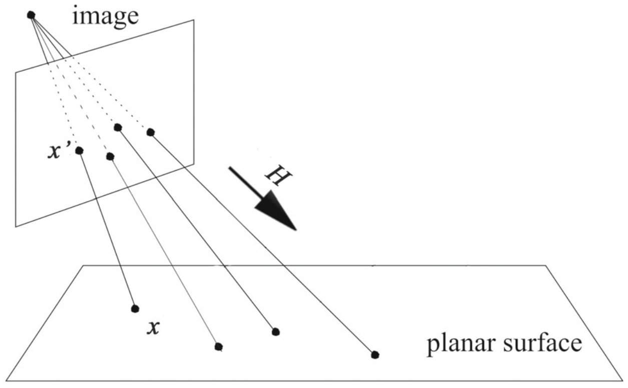

Fig. 2 A visual representation of homography and the corresponding points

audio-visual cues. It evaluates critical social density and modulates inflow into a region-of-interest.

Another work [12] defines social distancing monitoring as a visual social distancing problem. This work introduces a skeleton for a detection-based approach to inter-personal distance measuring. It also discusses the effect of social context on people's social distancing and raises privacy concerns. As a continuation of this work, [1] uses skeletons detection for estimating the homography by assuming parameters and estimates scales using average body part dimensions. This work does not address occlusions or any sort of tracking. It also does not address false social violations caused by people who are together such as family or friends. Another technique [7] localizes human posture estimations using only the intrinsic camera parameters and a machine learning model. This work seems to be useful when the camera is moving and not stationary, i.e., the extrinsic parameters cannot be determined.

3 Planar homography estimation

Homography is a special case of projective geometry. It enables the mapping of points in spaces with different dimensionality. A point x' observed in a view (such as an image) can be mapped into its corresponding point x in another perspective or coordinate system (such as a ground plane) (Fig. 2).

We can utilize this property to localize the pedestrians in view and calculate their locations on the ground with respect to each other. For this purpose, we need to find the homography matrix between ground plane in real-world coordinates and ground plane as it is represented in the image plane. This can be achieved by finding at least four corresponding points between the two planes and using them to compute the matrix elements.

The only restriction is that the four points must be in a "general position," which means that no three points are collinear. One thing to note here is the computation of the homography matrix H in this way does not require knowledge of any of the camera's parameters or the pose of the plane [18]. That being said, if correspondence cannot be established, but the intrinsic and extrinsic cameras parameters are known, the corresponding ground points can be computed by using a camera model [25].

There are also techniques available for automatic homography estimation using scene geometry [1] or machine learning [7]. These methods do not require any correspondence or extrinsic parameters. However, their reliability when it comes to accuracy is debatable. Therefore, it is recommended these techniques can be employed only as a last resort.

4 Pedestrian detection with YOLOv5

The primary objective in this step is to detect bounding boxes of pedestrians in the image frame. We chose YOLOv5 [19] for this task. Yolo is a fast and accurate neural network for object detection [29]. The Yolo architecture splits the input image into grids and for each grid generates bounding boxes and class probabilities for those bounding boxes. This type of model has several advantages over traditional classifier-based systems. It looks at the whole image at test time, so its predictions are informed by global context in the image. It also makes predictions with a single-network evaluation unlike systems like R-CNN which require thousands of passes for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN [29]. Yolo has had many improvements done to it over the years as newer versions come out. The YOLOv5 repository is a natural extension of the YOLOv3 PyTorch repository by Glenn Jocher. YOLOv5 is implemented natively in PyTorch, whereas all prior models in the YOLO family leverage Darknet.

The principal motivation behind the choice of YOLOv5 over v4 or v3 is speed. Since this implementation requires real-time inference, YOLOv5 has an advantage over its competition. YOLOv5 also provides us with flexibility and ease of use. We can pick and choose between speed and accuracy by leveraging its different network models (Fig. 3). "YOLOv5s" is the best when it comes to speed of inference, whereas 'YOLOV5m' is the optimal middle ground between speed and accuracy. Each frame is forwarded to the Yolo neural net as input and it outputs predictions with confidence levels as to what objects are present in the frame. All predictions below a threshold confidence level τ (0.5 in our case) are rejected. The locations of these predictions are returned as a center point ($c_{x'}$,$c_{y'}$) and height (h_b) and width (w_b) of an enclosing bounding box. Finally, the detections go through a process of non-maximal suppression to eliminate duplicates; Fig. 4 shows the final results of a frame.For our purposes the

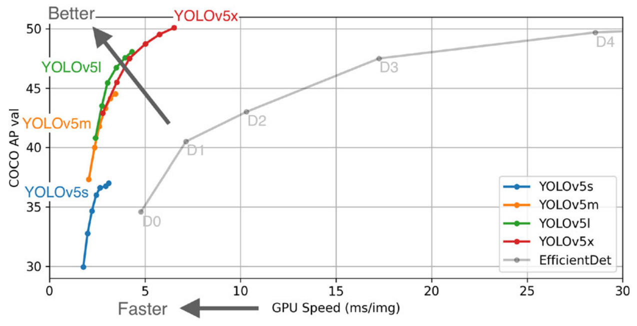

Fig. 3 Graph showcasing performance of different YOLOv5 models taken from Yolov5 GitHub repository

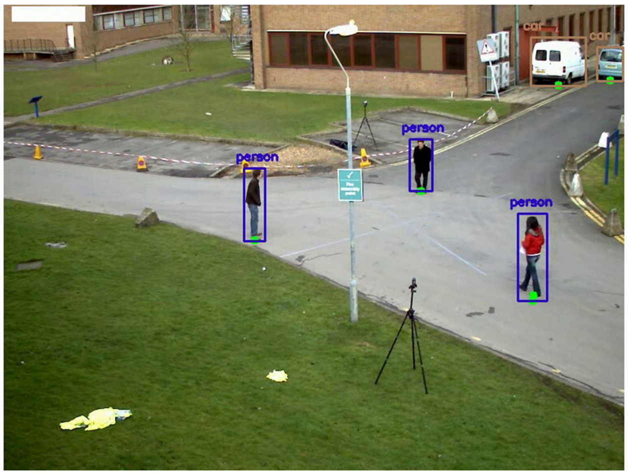

Fig. 4 Bounding boxes drawn on an original frame from the result of the YOLOv5 neural net

pre-trained models provided by the authors are adequate and perform well without any extra training.

5 Human pose estimation using OpenPifPaf with detection

For the estimation of poses we use a OpenPifPaf model pretrained on the COCO dataset. OpenPifPaf is a bottom-up method for multiperson 2D human pose estimation [20]. OpenPifPaf's architecture is based on a fully convolutional singleshot design. The architecture has two head networks. For each body part or joint, one head network predicts the confidence score, the precise location and the size of this joint, namely part intensity field (PIF). The other head network called the part association field (PAF) predicts associations between parts to form full human poses. OpenPifPaf's strength lies in its exceptional performance at low resolution and in crowded, and occluded scenes thanks to composite field PAF encoding fine-grained information and Laplace loss for regressions. OpenPifPaf specifically addresses challenges of limited resolution and high-density crowds where pedestrians occlude each other.

Although OpenPifPaf is meant as a bottom-up approach to pose estimation (estimating joints without a person detector), we do not use it standalone. We combine it with the YOLOv5 object detector to effectively implement it as a top-down method (i.e., using a person detector first and estimating joints within the detected bounding boxes).The main motivation behind this change is the methodology of training the OpenPifPaf model: using square image crops to create uniform batches and its scale-dependent adaptive regression loss [20]. Essentially, providing the model with images similar to training environment should, in theory, improve performance. This is because OpenPifPaf—despite its superior performance on low resolution, crowded, and cluttered scenes and scenes with occlusion—struggles when the resolution of the pedestrian is very poor (far away from the camera), which goes below 40 pixels height in our experiments. Fortunately, YOLOv5 has far better low-resolution performance and is adequately able to identify these pedestrians (Fig. 5). This combination thus results in greater speed and greater estimation of poses without further training in a pedestrian setting. Through our testing, we discovered that resizing the bounding boxes of the pedestrians into an aspect ratio of 1:1 or 1:2 improved detection of joints by the OpenPifPaf network on these very low resolutions.

These discoveries led to the following implementation: Firstly, the pedestrian bounding boxes are detected using an object detector (YOLOv5). Second, as we are assuming a pedestrian setting, most people are assumed to have an upright posture, i.e., standing, walking or running (although other poses perform adequately in our testing as well). With this heuristic in mind, all detected bounding boxes are cropped out from the image and then resized to a pixel size of 50×100 as shown in Fig. 6.

We then perform batch inference, using the OpenPifPaf network, on all these cropped detections as all are the same resolution. As a by-product of this batch inference the speed is dramatically increased, which is another major goal of this work. Thus, not only does this method improve detection of very low-resolution pedestrians, it also speeds up inference more than 5 times when measuring fps (frame per second). To demonstrate this, we run OpenPifPaf solo, and combinations of Yolov5s and OpenPifPaf, and Yolov5m and OpenPifPaf, on the PETS dataset [13]. The pretrained backbone used for all testing and implementation is shufflenet. The number of skeletons detected and average FPS is recorded on the PETS 2009 s2,L1 View 001 and is displayed in Table 1. To evaluate the performance on this dataset, the ground truth is calculated by the manually counting the number of individuals visible in each frame. Our method results in more detections and up to 5x higher fps on lower end graphics hardware. Examples from this test can also be seen in Fig. 5

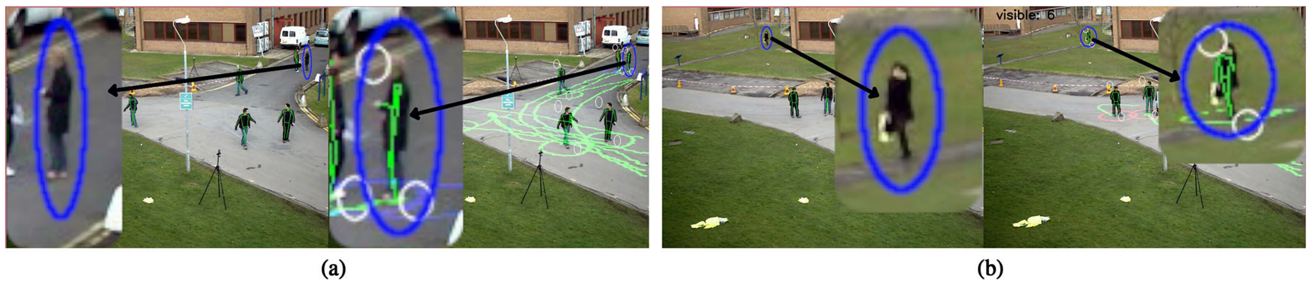

Fig. 5 Two examples of how our system is able to detect very low-resolution skeletons where solo OpenPifPaf fails. In both (**a**) and (**b**) results with solo OpenPifPaf are shown on the left and results with our system on the right

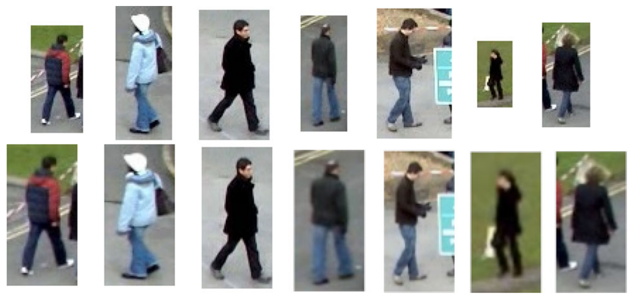

Fig. 6 All detected bounding boxes cropped out (top) and then resized to 50 × 100 (bottom)

where the pose of women in the background is not detected when only OpenPifPaf is used. They are, however, detected when the combination of the two is used. This can also be used for training the stand-alone OpenPifPaf network, further improving its accuracy as manually labeling poses can be very expensive. As the skeletons were counted manually frame by frame and there are no ground truth annotations, thus, accuracy of the labels cannot be judged by this test. We further evaluate accuracy in validation listed below.

Our system is also validated on 100 images picked randomly from the CrowdPose dataset [22] with the only limitation being the image should be somewhat similar to a pedestrian setting, i.e., people are standing, walking or running. For consistency with OpenPifPaf, evaluation is done using the COCO API. As CrowdPose annotations use different keypoints for the head and neck when compared to COCO, to keep the testing consistent, these are not used when validating. The results are shown in Table 2. As ground truth labels in the original dataset do not include labels for pedestrians that are farther away from the camera, these annotations are added by us to the ground truth using COCO annotator [9]. These updated annotations will be released along side the code of this work. For completeness, we also compare with a SOTA technique DEKR [14]. We use DEKR with HRNet-W32 backbone pretrained on the COCO dataset by its authors. The average precision of our technique is higher when compared with OpenPifPaf solo, whereas average recall is higher across the board. The higher recall results from our model being able to estimate poses for pedestrians very far away from the camera. These are not even predicted by DEKR despite slower inference speed. Figure 8 shows an example. Average precision performance with a 50 percent overlap is superior for our model. Lower performance in the 75 to 95 percent overlap precision could be attributed to resize losses. Some examples are shown in Fig. 7.

An unwanted side effect of our method is when bounding boxes overlap there can be duplicate pose estimations. However, this can easily be remedied, with minimal hit to performance, by comparing the mean of Euclidean distances between corresponding joints of the estimations in these overlapping bounding boxes. If An_i is a set joint coordinates estimated to be corresponding to a single pedestrian, such that:

$$An_i = \{(x_{i,1}, y_{i,1}), (x_{i,2}, y_{i,2}), ...(x_{i,17}, y_{i,17})\} \quad (1)$$

then two annotation sets An_j and An_k from overlapping bounding boxes can be compared as follows:

Table 1 Comparison between OpenPifPaf solo and our method on a system with a GTX 1050tiM and core i7-8750H on the PETS 2009 s2, L1 View 001 [13]

Methods	Skeletons	Bounding box	fps
Total (ground truth)	5020	5020	–
OpenPifPaf solo	4498	4498	4.6
Our method (with YOLOv5s)	4504	4556	**22.3**
Our method (with YOLOv5m)	**4563**	**4679**	15.6

The number of skeletons estimated and average FPS is recorded and displayed
The bold specifies better detection of skeletons and bounding boxes from frames by our method when compared with OpenPifPaf. While the last column specifies best FPS

Table 2 Validation results on 100 pedestrian setting images from the CrowdPose dataset [22], after revising the validation set to include missed labels

Method	Average precision (AP)			Average recall (AR)			FPS
	IoU=0.50:0.95	IoU=0.75	IoU=0.50	IoU=0.50:0.95	IoU=0.75	IoU=0.50	
OpenPifPaf (solo)	0.421	0.451	0.625	0.573	0.613	0.722	2.85
Our method (with YOLOv5x)	0.451	0.470	0.652	**0.644**	0.676	**0.797**	2.87
Our method (with YOLOv5l)	0.441	0.467	0.633	0.638	0.677	0.789	3.45
Our method (with YOLOv5m)	0.458	0.491	**0.667**	0.640	**0.680**	0.792	4
Our method (with YOLOv5s)	0.455	0.493	0.650	0.623	0.669	0.762	**5**
DEKR	**0.515**	**0.585**	0.652	0.593	0.644	0.680	1.66

Crops are resized to 100 × 200
The bold specifies the best performance in the column

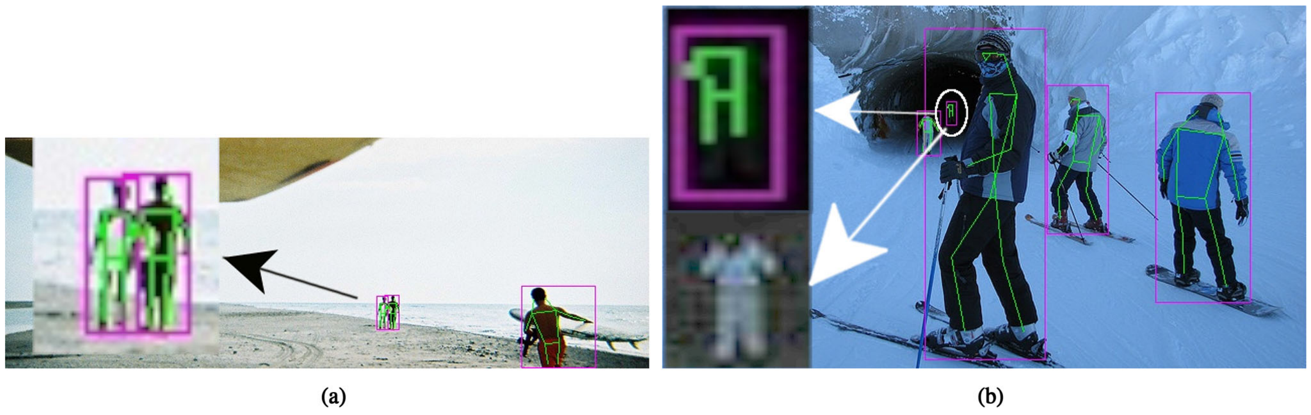

Fig. 7 Two examples (shown in (**a**) and (**b**)) of how our system is able to detect very low-resolution skeletons on the CrowdPose dataset [22]. In (**b**), an overexposed crop of the pedestrian in the dark tunnel is also shown, which is very hard to see in the regular image

$$X = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(x_{j,i} - x_{k,i})^2 + (y_{j,i} - y_{k,i})^2} \qquad (2)$$

If X is less than a threshold τ (=5 in our case) , the duplicate with fewer keypoints is discarded. We also implement non-maximal suppression from [10] as an alternative to remove duplicates. Users have the option to choose between them. The resize resolution of 50 × 100 can also be increased, maintaining aspect ratio, if higher accuracy is preferred over speed.

6 Tracking using DeepSORT with optical flow

For tracking we use a system similar to FASTMOT [36], but with modifications made to the object detector, feature extractor, tracking system and how the optical flow is Incorporated. Also changes were made to make the system compatible with Pytorch library. This system has MOTA scores close to state-of-the-art trackers (Table 3) while still being able to run in real time. Multi-Object Tracking Accuracy (MOTA) is the standard metric used to evaluate multi-object tracker systems [6]. In this section we will focus on how the tracking system works, as at the time of writing there is no paper published to support the work in [36] and also go in to the detail of the modifications we made to improve the system for our implementation.

The system core function in [36] is similar to the DeepSORT algorithm in [34]. Optical flow is added in order to improve tracking accuracy, camera motion detection and increase fps performance by skipping running the detector on intermediary frames using optical flow prediction to fill in the gaps. We replace the object detector with YOLOv5 which results in a drastic speed improvement. We also abandon the camera motion detection as CCTV camera is assumed to be stationary, and therefore we avoid this unnecessary overhead.

As stated earlier the tracking system functions similarly to the DeepSORT algorithm. The bounding boxes from the object detector are used to initiate tracks and extract features. One key difference from DeepSORT is that the feature extractor neural net is replaced by OSNET REID [37]. Dubbed the omni-scale network (OSNET) by its authors, its architecture consists of a residual block composed of multiple convolutional streams, each detecting features at a certain scale. This type of design allows it to capture features of both homo-

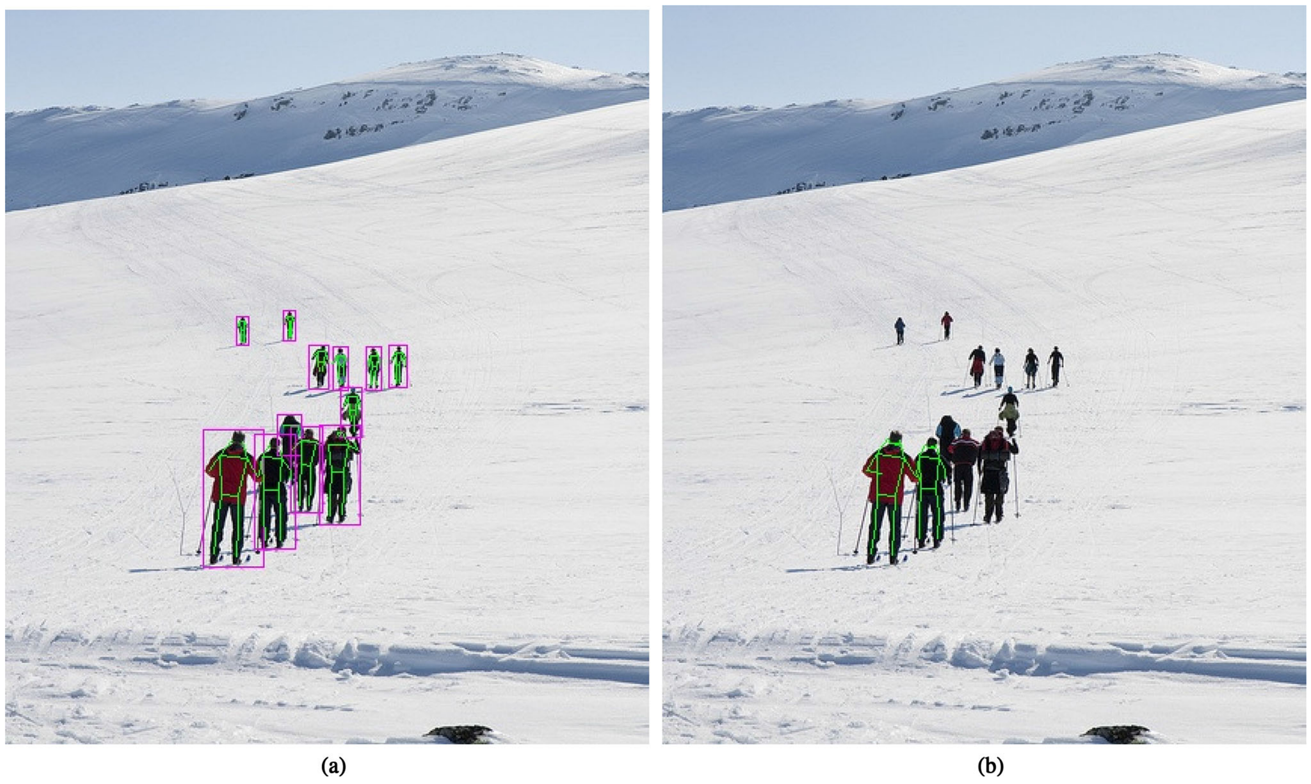

Fig. 8 A comparison from the CrowdPose dataset showcasing superior performance of our method on far away pedestrians: **a** shows output from our method (YOLOv5 bounding box and OpenPifPaf pose estimation); **b** shows output from DEKR [14]

Table 3 FASTMOT on MOT20 train set

MOTA	IDF1	HOTA	MOTP	MT	MLs
66.8%	56.4%	45.0%	79.3%	912	274

Taken from [36]

geneous and heterogeneous scales. A unified aggregation gate is employed to then dynamically fuse multi-scale features with input-dependent channel-wise weights. By using point-wise and depth-wise convolutions, the model is able to efficiently learn spatial correlations and avoid over-fitting. In addition, OSNET is extremely lightweight. This allows inference in real time, and despite its small model size, it achieves state-of-the-art performance outperforming most large-sized models. In our implementation the model generates a 512 feature set from a bounding box which is bespoke to each person and can be used for tracking along with re-identification after a track is lost or goes out of frame.

With each successive re-identification of a track, the feature set is updated fractionally by the newly extracted feature set by a factor θ. If f_{old} are the old features and f_{ext} are the newly extracted feature from the current frame, then the new feature set for the track f_{new} would be:

$$f_{\text{new}} = f_{\text{old}} * (1 - \theta) + f_{\text{ext}} * \theta \tag{3}$$

Now we discuss how optical flow is incorporated into the algorithm. At initialization, image frame is converted into gray scale. Then corners inside each bounding box are detected using OpenCV's implementation of the Shi-Tomasi corner detection method [30] to find the strongest corner points. These points are then passed through an ellipsoid filter. This filters out all keypoints not inside an ellipsoid drawn with the bounding box center as its center, and height and width of the bounding box as its major and minor axes. This is done to remove points at the edges of the bounding box. The remaining points are to be used as keypoints for optical-flow calculation. This step is only performed when none or too few previously predicted keypoints are present in the current bounding box. Otherwise the predicted keypoints are recycled.

After keypoint detection, their motion in the current frame is predicted using OpenCV's implementation of the Lucas-Kanade Optical Flow [23], trying to find their matches in the current frame. All points without matches or those that exceed max error threshold (100 in our case) are dropped. A partial affine transformation matrix between the previous and current matched points is computed. This matrix is used to transform the current bounding box to the optical flow prediction of its position in the next frame. This prediction is used to update the Kalman filter in the DeepSORT algorithm.

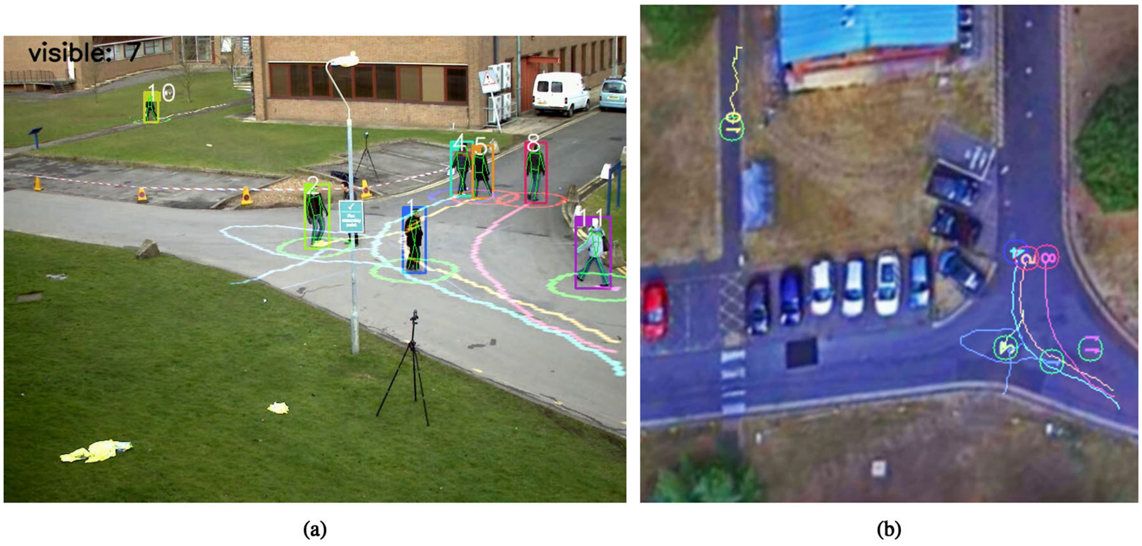

Fig. 9 Tracking using our system described in Sect. 6: **a** paths drawn corresponding to the tracks. **b** A top-down view

Thus the Kalman filter receives updates twice per frame, one from the optical flow prediction (giving higher uncertainty to the optical flow prediction) and the other when the track is updated with the final match.

The bounding-box matching is done identically to the DeepSORT algorithm. First, the Mahalanobis distance between the Kalman filter predictions and detected bounding boxes and Euclidean distance between the feature embeddings are calculated and then fused together into a single cost using a weighting metric. The distances are also checked to see whether they are within the gating region for both metrics. A linear assignment algorithm is used to match detected bounding boxes to the tracks. Then those unmatched are matched using IOU cost, i.e., how much the bounding boxes overlap and if it is over a certain threshold (over 40 percent overlap in our case), giving priority to those that have previously been matched. Finally the tracks left unmatched are analyzed for re-identification with previously lost tracks, whether the extracted feature embeddings distance is below a certain threshold (0.6 in our case). This works as people exiting and then reappearing in the frame or coming out of occlusion will almost always not match with any other track. An example of our tracking system can be seen in Fig. 9. The visualization technique used in this figure will be described in Sect. 9.

7 Pedestrian localization and distance calculation

As previously discussed in Sect. 3, we need to find the piercing point of the pedestrians with the ground plane in order to localize them on the ground using homography. The skeletons of each pedestrian detected inside their respective bounding boxes are tracked using our tracking method described in Sect. 6. This allows us to not only infer the piercing point of each pedestrian, on the image, using their feet but also if the feet are occluded, we can infer the piercing point using the dimensions and location of other estimated body parts or using previous skeleton estimations of the track (Fig. 10). Note that if only the bounding boxes were detected, the system would be unable to deal with occlusions, or rather it would be unaware if an occlusion has even occurred. Since it would have no information as to how much of the body is captured inside the bounding box. This is the main advantage of our method over other more trivial implementations [12,27].

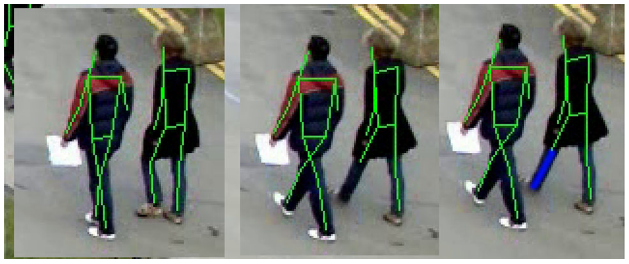

Fig. 10 Left shows the previous frame where the complete leg is detected. Middle shows the current frame where the lower leg is not detected by OpenPifPaf. Right shows the lower leg estimation in the current frame using our pedestrian localization technique

In the optimal scenario where there are no occlusions, the piercing point is estimated as the center point between the two feet of a pedestrian. As all body parts detected by the pose estimator are labelled as such, if both or either one of the feet is occluded, but the thigh is visible and detected, we assume the rest of the leg to be a natural extension and thus extract the dimension of the lower leg from the previous

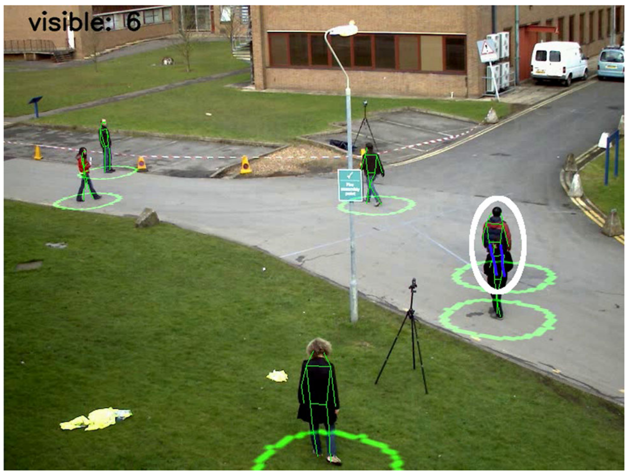

Fig. 11 In the case that only torso is detected (for the first person on the right of the image who is occluded by the person behind), both legs are reconstructed from previous skeletons of the track

skeleton estimations of the track and add it to the end of the thigh using the thigh's slope as the slope for the lower leg. If there is no previous skeleton, we assume symmetry and extract the dimension of the other lower leg and append it to the end of the thigh, thus completing the skeleton. We want to note here that the machine learning model OpenPifPaf's output has labels on which body part is detected. Therefore, even if the upper body is not detected, the thigh (i.e., the upper part of the leg) can be identified by the network. If m is the slope of the thigh and (x_t, y_t) are the coordinates for the knee(thigh endpoint) and d is the extracted dimension of the lower leg, then we can estimate the coordinate for the corresponding foot F as follows:

$$F = \left(x_t + d.\sqrt{\frac{1}{1+m^2}}, y_t + d.m.\sqrt{\frac{1}{1+m^2}} \right) \quad (4)$$

If the complete leg is occluded, i.e., even the thigh is not detected, we look for the lower end of the torso or corresponding side of the torso as shown in Fig. 11. If it is detected, we can still infer the leg positions in a similar fashion. Even if none of the above-mentioned resources are available, we can still use estimations based on the average human proportion distributions in the camera feed to infer the piercing point. As the distribution is localized to each camera, the effect of the camera angle skewing proportions is minimized. Thus a piercing point can be established to a reasonable accuracy even if only the head of the pedestrian is free of occlusions. If the skeleton cannot be estimated from the image but the bounding box is detected, we estimate the piercing point as the mid-point of the bottom side of the bounding box as a fallback. We also look at the track history to estimate average size of pedestrian to prevent abrupt changes to bounding box size and improve piercing point accuracy. Finally, the piercing point can be transformed into real-world coordinates (p_x, p_y) using the homography matrix H calculated in Sect. 3.

The knowledge of the coordinates of all pedestrians in the image with respect to the ground plane in the real world reduces the problem of calculating distance between them to Euclidean distance between two points in 2D plane. Regardless of the coordinate system selected on the ground plane, we can find the distance between any two pedestrians by calculating the Euclidean distance [32] between their points of intersections on the ground plane as

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (5)$$

where d_{ij} is the distance between any two pedestrians having piercing points (x_i, y_i) and (x_j, y_j) on the ground.

8 Smart violations detector

When calculating whether there has been a social distancing violation, we also want to take into account detecting whether two or more pedestrians are together. For example, family members walking together would not be considered a social distancing violation. For this reason, we developed a "smart" social distancing measure technique for estimating if pedestrians are together.

Walking or standing. We first detect and segregate pedestrians into two categories: walking or standing. If the pedestrians are deemed walking, then the Fréchet distance between their tracks for a predetermined amount of time is computed. The Fréchet distance is used to measure similarity between curves taking into account the location and ordering of the points along the curves [4]. The ordering part is useful as we measure max distance between pedestrians relative to time; what position they both were at that specific time or on that specific frame. If it is less than a certain threshold, then these two pedestrians are deemed together. We use 1.5 meters as the threshold. Using this technique we are able to detect even if multiple people are together and assign them to the same group (family or friends) (Fig. 12). These groups are assumed together only within the group and any violation with a pedestrian or group outside the current group is detected as such. For weeding out crowds, any group of people over 6 is automatically removed of its assignment as a group and considered violating social distancing. This number can be programmatically altered if desired.The system is limited in the fact that it lacks semantic understanding. When people are walking together due to crowded areas or narrow walkways, in such cases only a semantic understanding of the scene can help evaluate if two people are together. As no dataset is currently available for this task, training a model

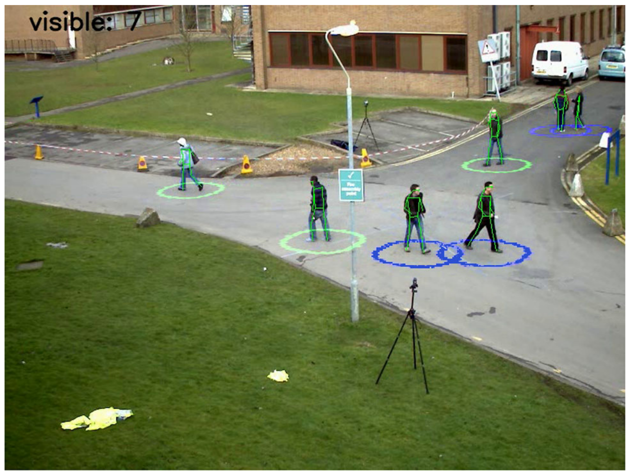

Fig. 12 People walking and having Fréchet distance of their paths less than a threshold are considered together and have blue circles around them

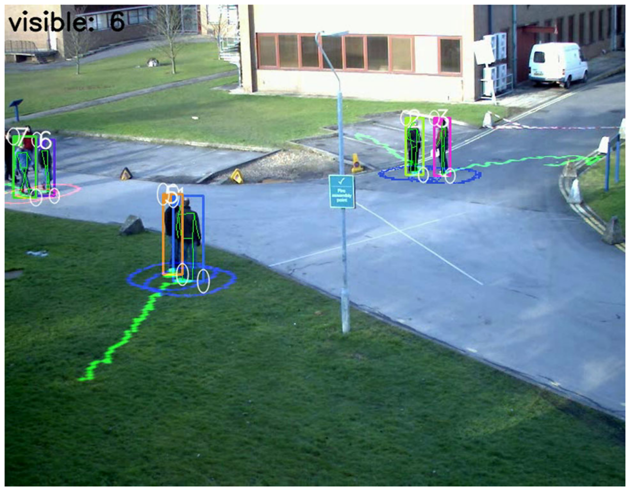

Fig. 13 People standing and facing each other are considered together and have blue circles around them

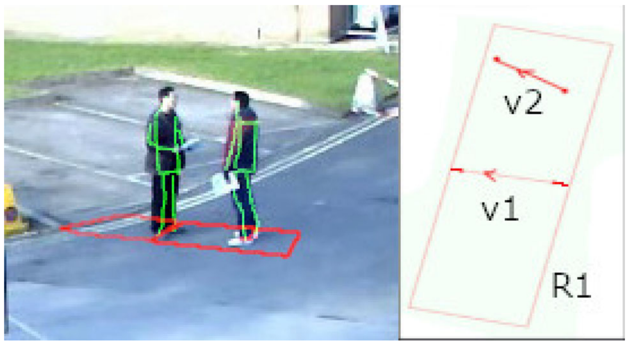

Fig. 14 The vector between the legs of each person is obtained (v_1, v_2) and the relation between the these vectors is used to figure out if they are standing in front or behind each other

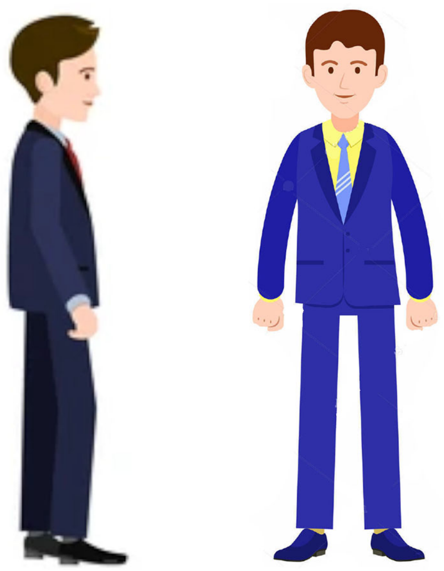

Fig. 15 People standing in front of each other but not facing each other. The angle between v_1 and v_2 is close to 90 degrees

is not possible. Some fail-safes are put in place to somewhat remedy this including the crowd limit stated above or the tweaking the together-distance threshold.

People are estimated as standing if they do not move for a predetermined amount of time. When standing two people are considered to be together only if they are facing each other. As it makes sense for family or friends standing together to interact with each other.

Facing each other. Detecting if people are facing each other without 3D skeletons can be a daunting task. Hence, we use heuristics to determine whether two people are facing each other. First, a vector v_1 is drawn from the left to the right foot coordinates localized on the ground plane, of one of the pedestrians. The slope of v_1 is used to draw a rectangle R_1 where the v_1 is the line passing through the center of the rectangle as shown in Fig. 14). The width of R_1 is equal to the width of v_1 and length is set to 3m (i.e., 1.5m each side). We then check if one of the feet of the other pedestrian lies inside this rectangle or the vector v_2 between the feet (left to right) of the other pedestrian intersects this rectangle and angle θ_1 between v_1 and v_2 is not close to 90 degrees. The reason for checking the angle is to see if the person is standing facing toward or away and not at a 90 degree angle as shown in Fig. 15. If this check is validated, we can move to the next step. Now we have determined that the other person is standing in front or behind of the first pedestrian. When pedestrians are facing each other, they can be viewed in frame either as one pedestrian facing the camera and the other facing away or both can be seen from a side view. An example can be seen in Fig. 13. In the first case, we check for both or either eyes detected for one person and only shoulders and/or ear detected for the other to validate if they are facing each other. To demonstrate this in Algorithm 1,

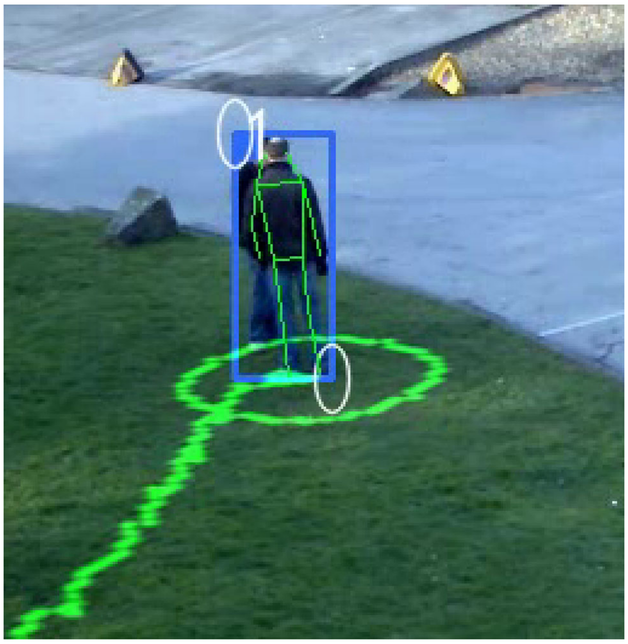

Fig. 16 Limited detection if people are together when complete occlusion occurs

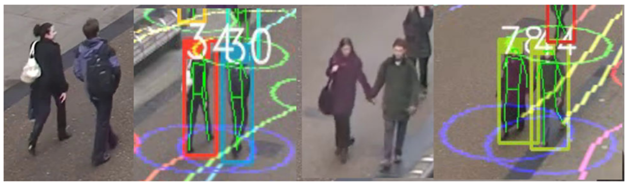

Fig. 17 People seen talking with each other and holding hands are detected as together by the smart violations detector

we use Booleans e_l1, e_r1, e_l2 and e_r2 which correspond to the left and right eyes of the first pedestrian and left and right eyes of the second pedestrian, respectively. These Booleans are true when the corresponding eyes are visible in frame. We also employ two other Booleans ch_1 and ch_2 to check if either of the shoulders or ears are visible, one Boolean for each pedestrian, respectively, to tackle occlusions. If both eyes for both pedestrians are detected or vice versa that means they are facing the same direction and thus not facing each other.

For the second scenario, we measure the distance between the eyes/ear/nose of one pedestrian to the eye/ear/nose of the other pedestrian. If looking at each other the distance from the eyes of one pedestrian to the eyes of the other pedestrian should be smaller than to the ears of the other pedestrian and vice versa if facing opposite or the same direction. A final test for robustness is put in which is optional and can be omitted: we measure the angle θ_2 of the vector between the ears and eyes for both pedestrians. If both are facing the same direction, it will be less than 90 degrees.

Algorithm 1 Check if two people are facing each other when standing

```
Require: $v_2$ is inside or intersecting $R_1$
Ensure: $\theta_1$ is not close to 90
  if any($e_{l1}$,$e_{r1}$) and (not all($e_{l2}$,$e_{r2}$) and $ch_2$) then
      return True
  else if any($e_{l2}$,$e_{r2}$) and (not all($e_{l1}$,$e_{r1}$) and $ch_1$) then
      return True
  else if any($e_{l2}$,$e_{r2}$) and any($e_{l1}$,$e_{r1}$) and (not all($e_{l2}$,$e_{r2}$) or not all($e_{l1}$,$e_{r1}$)) then
      if $\theta_2 > 90$ and dist($eye_1$,$ear_2$)> dist($eye_1$,$eye_2$) then   ▷ $\theta_2$ check is optional
          return True
      end if
  end if
  Return False
```

We do not actually need to know which scenario the pedestrians are standing in as the checks are sequenced to work for both scenarios. The standing methodology is limited in cases of severe occlusions for example, when one of the pedestrians is completely occluded by the other in the frame as shown in Fig. 16 .

Evaluation of togetherness. The social distancing together groups are reevaluated every second. For evaluation of the system on a real-world scenario we employ Oxford Town Centre dataset [5]. This dataset is a CCTV video of pedestrians in a busy downtown area in Oxford. We calculate homography without camera parameters by employing Google Earth [15] and using bench locations visible in the camera and on Google Earth to collect corresponding points. We were able to obtain under 30 cm of error when localizing pedestrians on the walkway.

As no ground truth is available for people being together we only look at people holding hands or interacting with each other (mostly talking to each other) to establish ground truth, Fig. 17. The smart violations detector is able to categorize these pedestrians together 81% of the time (17 out of 21), as shown in Table 4. Only 2 times the detector categorized people together which were not picked as convincingly together in the ground truth. The smart violations detector misses cases where the object detector fails to consistently detect a pedestrian due to constant occlusions or the pedestrians are not in frame long enough for the necessary amount of localization data required.

We also loop the dataset and have the system running for 2 hours on our machine to test robustness in a streaming scenario. The footage is resized from 1920× 1080 to 1280× 720, and a frame skip is employed processing 5 out of the 25 frames a second in order to maintain real-time operation on our machine. A two second buffer is provided. The results are shown in Table 5. The system is able to maintain consistent frame rate, and detections remain consistent with frames on subsequent loops.

Table 4 Total together detections from the Oxford Town Centre dataset [5] that are interacting with each other

Ground truth (GT)	Total detected	In GT	Not in GT
21	19	17	2

9 Social distancing visualization

The distances need to be represented in an intuitive way that can be easily interpreted and is not overwhelming to the viewer. Initially, the raw representation was adding green distance lines between pedestrians if the distance was less than 5 meters (16.4 feet) and displaying the distance on top of the lines. These lines would turn red if the distance went below 2 meters or roughly 6 feet (Fig. 18a). With blue lines representing pedestrians that were together, this representation becomes overwhelming and confusing if there are many pedestrians in view (Fig. 18b).

For an image-based visualization, the best representation we could come up with was green circles of a 3-foot radius around each pedestrian. When any two circles intersect it can be interpreted as the two pedestrians are closer than 6 feet and the circles turn red alerting the viewer. Similarly blue circles to represent people that were together. Now the challenge we faced implementing this was drawing circles in real time that would correctly reflect the distance in the real world on the image. As drawing circles of fixed size would not show the distance accurately. Also, to make it work in real time transforming each point of a circle before drawing it on the image was not computationally viable.

Therefore, what we ended up creating was a separate blank image which represented a scaled top-down view of the real-world ground plane. This blank image would act as a buffer between the ground plane and image. The reasons for scaling the blank image were twofold: so that all of the ground plane coordinates visible in the camera feed were represented as positive pixel values on the blank image and the blank image size would not be excessive. The circles and the moving average of the tracks (to smooth them out) were then drawn on this blank image (Fig. 19) and transformed using homography and overlaid on to the original image using the following relations.

A point P_i on the image of the ground plane could be represented as:

$$P_i = H_{ip}.P_p \tag{6}$$

where P_p is the point on the ground plane in real-world coordinates and H_{ip} is the projective transformation or homography between the two planes. The same point on the blank image P_b can be represented as

$$P_b = H_{bp}.P_p \tag{7}$$

where P_b is the point on the blank image and H_{bp} is the projective transformation between real-world ground plane and blank image (Fig. 20).
Now Eq. (7) can be written as:

$$P_p = H_{bp}^{-1}.P_b \tag{8}$$

Putting value of P_p in Eq. (6) from Eq. (8) we get:

$$P_i = H_{ip}.H_{bp}^{-1}.P_b \tag{9}$$

Thus, by using this relation each point in the blank image is mapped onto the camera frame. And since the blank image is just a scaled representation of the real-world coordinates, this resulted in circles which were correctly scaled (Fig. 21a). The blank image in Fig. 21b is also used in the result to represent the top-down view of the ground plane.

10 Implementation details and results

This work is implemented on the PETS 2009 s2, L1 View 001. We use a calibrated camera with calibration parameters already provided in the data set to find corresponding points for the purpose of computing homography. That being said, it is not necessary for the camera to be calibrated for this work to be implemented [18]. To compute planar homography only 4 corresponding points between 2 planes are required as already explained in Sect. 3.

The application is run on a Intel core i7-8750H CPU and a Nvidia GTX 1050tiM GPU and 16GB RAM. The CCTV

Table 5 CPU, GPU,RAM usage and fps reported on the Oxford Town Centre dataset [5]

Component	Average	Max
CPU utilization	44%	72%
GPU utilization	18.48% (747MB)	18.48% (747MB)
RAM utilization	2GB	2GB
FPS	5	7
Pedestrians visible in frame	15	24

The specifications of the machine are GPU:GTX 1050tiM and CPU:core i7-8750H

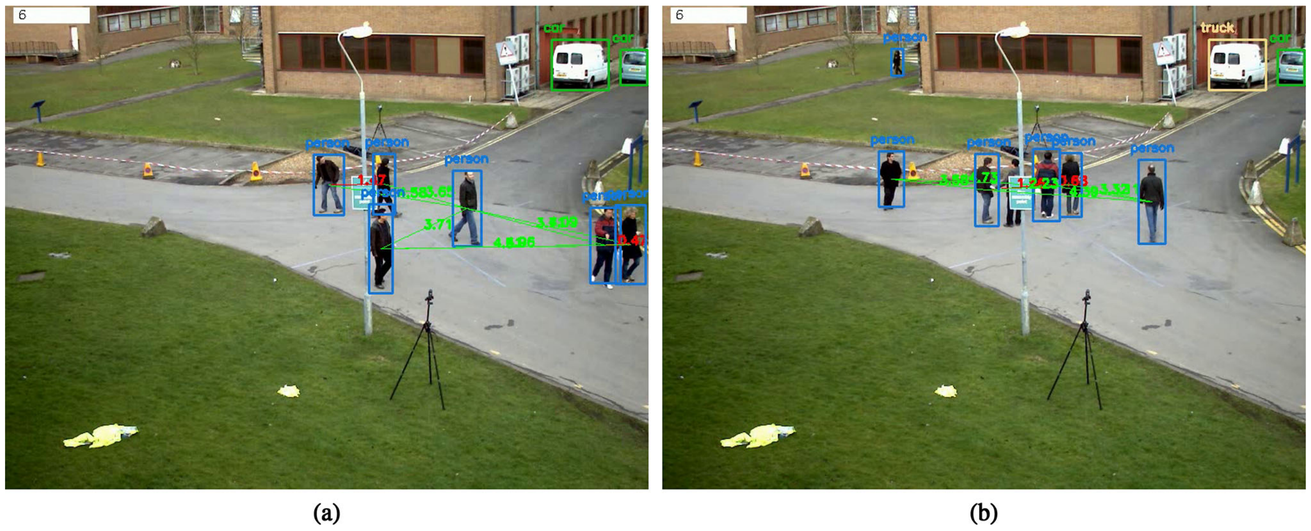

Fig. 18 Social distancing initial representation with bounding boxes and lines: **a** Bounding boxes for pedestrians and lines between them in green (greater than 6 feet) and red (less than 6 feet). **b** Dense crowds may overwhelm the viewer in this kind of view

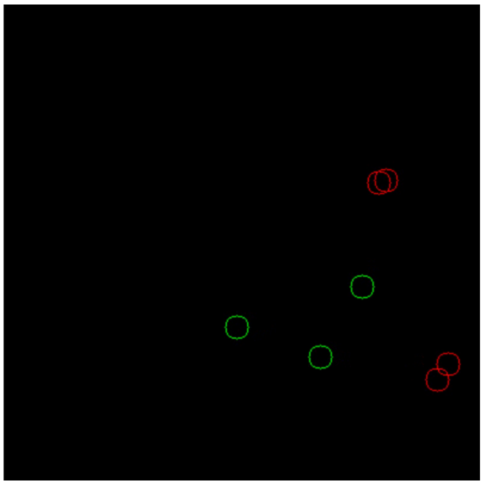

Fig. 19 Scaled blank image top-down view

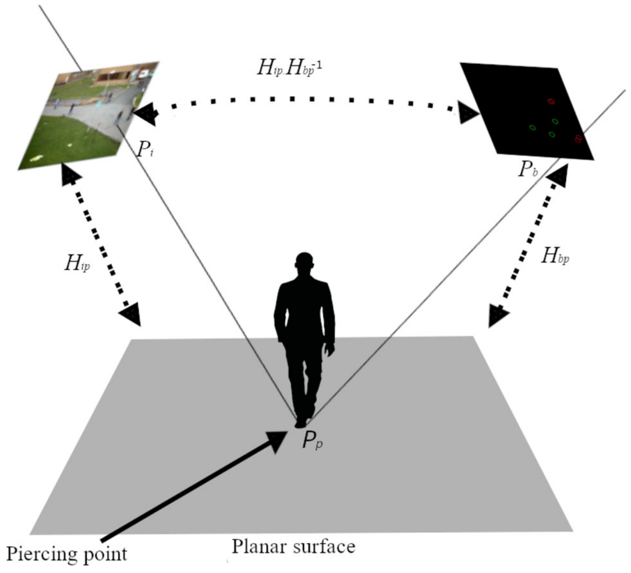

Fig. 20 The homography between two images induced by a world plane (the concatenation of two homography matrices)

video feed has 7 fps; our system runs at 9fps on this hardware. This results in smooth real-time operation. A video demo of the results of our work can be seen on YouTube at the following link: https://youtu.be/OUjvAlYy_vs. The videos show a top-down view and a regular view with overlay of the system's results as shown in Fig. 21. The blank image of the top-down view has the satellite image of the scene added to it courtesy of Google Earth [15] for visual appeal (Fig. 21b, Table 6).

11 Conclusion and discussion

We have presented a novel system called PETL4SD for tracking posture skeletons in real time in a pedestrian setting using CCTV cameras. We then employ this technique to detect and visualize social distancing violations. The pedestrian detection was performed using YOLO neural net. The posture estimation was done using OpenPifPaf. We validated our pose estimation on the CrowdPose dataset and the PETS 2009 dataset. Our system demonstrated superior pose estimation performance with no additional training. Tracking was performed using a modified implementation of [36]. The localization, relative distance and realistic circular overlays are achieved using planar homography and image composition (background, circles, pedestrians). All this sophistication leads to better social distance tracking

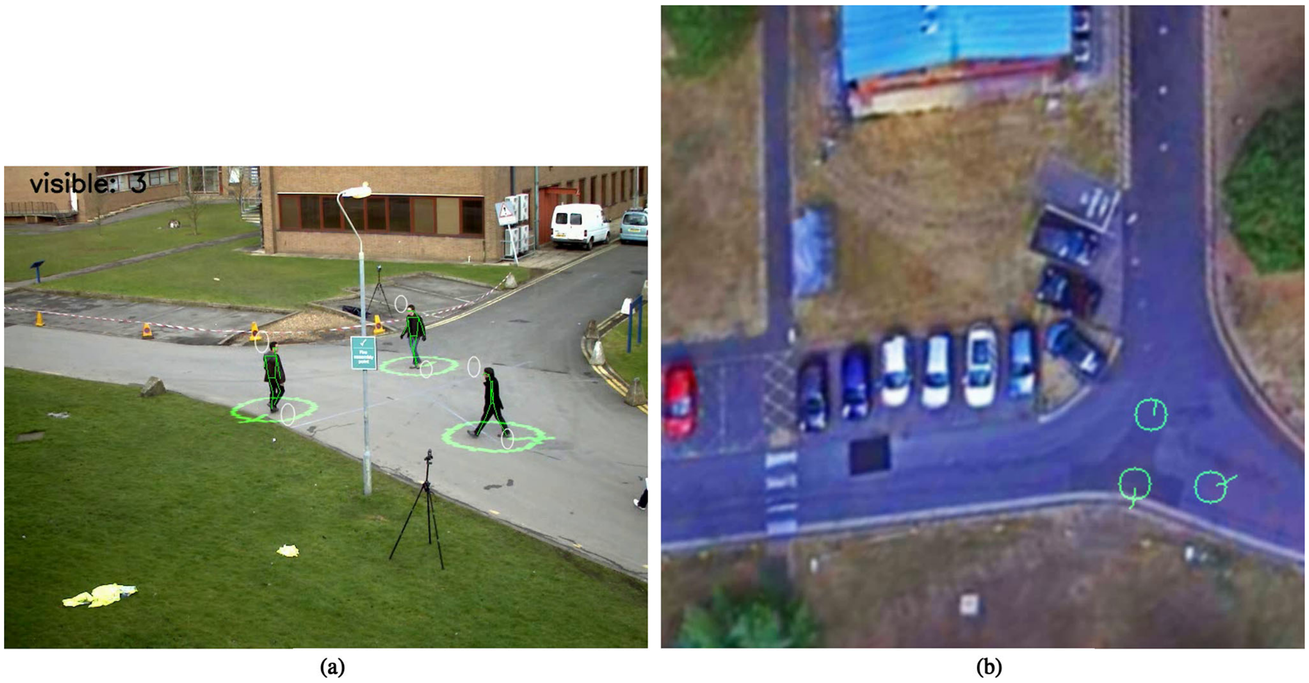

Fig. 21 Social distancing final representation. **a** Social distancing enhanced visualization with circles around pedestrians. The image is a real-time composition of background, circles and pedestrian images, showing the right perspectives, sizes and occlusions. **b** Top down view with the social distancing circles added to the corresponding satellite image

Table 6 Breakdown of the total time taken by each module when processing a frame

Module	Percentage of total time
Object detector (yolov5)	14 %
Pose estimation (OpenPifPaf)	55 %
Tracking+optical flow (FastMOT)	28 %
Social distancing (smart violations detector)	<1 %
Visualization	2 %

and monitoring when compared with more trivial systems, with no extra infrastructure and minimal requirements for initial setup. We also introduce a novel technique of detecting whether pedestrians are together and therefore negating false violations. We also evaluate this technique on a real-world CCTV dataset: Oxford Town Centre dataset. Great cares have been taken to optimize the time and detection performance of all the components; therefore, this work in its current form can be implemented for real-world use.

Nevertheless, the performance and tracking can be further enhanced by adding multiple camera angles to deal with occlusions. Using panoptic segmentation of the ground plane and estimates of average human dimensions and average walking speed we can develop a system to estimate homography from a video feed, thus eliminating the need for manual correspondence. With ground positions known we can estimate the homography for multiple imaginary planes with regular intervals across the video feed using the human body dimension estimates and, therefore, with some heuristics, be able to recreate a 3d scene from the CCTV camera feed. We can also employ the posture tracking data to train an action recognition neural net for other applications, for example, to infer fall detection, hostile behavior, etc.

Acknowledgements The work is supported in part by NSF via the Partnerships for Innovation Program (Award #1827505) and the CISE-MSI Program (Award #1737533), AFOSR Dynamic Data Driven Applications Systems (Award #FA9550-21-1-0082) and ODNI via the Intelligence Community Center for Academic Excellence (IC CAE) at Rutgers University (Awards #HHM402-19-1-0003 and #HHM402-18-1-0007).

References

1. Aghaei, M., Bustreo, M., Wang, Y., Bailo, G., Morerio, P., Bue, A.D.: Single image human proxemics estimation for visual social distancing. arXiv:1905.00953 (2020)

2. Ahmed, I., Ahmad, M., Rodrigues, J.J., Jeon, G., Din, S.: A deep learning-based social distance monitoring framework for covid-19. Sustain. Cities Soc. **65**, 102571 (2021). https://doi.org/10.1016/j.scs.2020.102571, https://www.sciencedirect.com/science/article/pii/S2210670720307897
3. Alhmiedat, T., Aborokbah, M.: Social distance monitoring approach using wearable smart tags. Electronics (2021). https://doi.org/10.3390/electronics10192435
4. Alt, H., Godau, M.: Computing the fréchet distance between two polygonal curves. Int. J. Comput. Geom. Appl. **05**(01–02), 75–91 (1995). https://doi.org/10.1142/s0218195995000064
5. Benfold, B., Reid, I.: Stable multi-target tracking in real-time surveillance video. In: CVPR 2011, pp. 3457–3464 (2011). https://doi.org/10.1109/CVPR.2011.5995667
6. Bernardin, K., Elbs, A., Stiefelhagen, R.: Multiple object tracking performance metrics and evaluation in a smart room environment. In: Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV, vol. 90. Citeseer (2006)
7. Bertoni, L., Kreiss, S., Alahi, A.: Monoloco: Monocular 3d pedestrian localization and uncertainty estimation. CoRR (2019). arXiv:1906.06059
8. Bradski, G.: The OpenCV Library. Dr. Dobb's J. Softw. Tools (2000)
9. Brooks, J.: COCO Annotator. https://github.com/jsbroks/coco-annotator/ (2019)
10. Cao, Z., Hidalgo, G., Simon, T., Wei, S., Sheikh, Y.: Openpose: Realtime multi-person 2d pose estimation using part affinity fields. CoRR (2018). arXiv:1812.08008
11. CDC: COVID-19 and Your Health (2020). https://www.cdc.gov/coronavirus/2019-ncov/transmission/variant.html
12. Cristani, M., Del Bue, A., Murino, V., Setti, F., Vinciarelli, A.: The visual social distancing problem. In: IEEE Access (2020). https://doi.org/10.1109/ACCESS.2020.3008370
13. Ferryman, J., Shahrokni, A.: Pets2009: Dataset and challenge. In: 2009 Twelfth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, pp. 1–6 (2009). https://doi.org/10.1109/PETS-WINTER.2009.5399556
14. Geng, Z., Sun, K., Xiao, B., Zhang, Z., Wang, J.: Bottom-up human pose estimation via disentangled keypoint regression. CoRR (2021). arXiv:2104.02300
15. Google earth. https://earth.google.com/
16. Hall, J.: Social distance monitoring (2020). https://levelfivesupplies.com/social-distance-monitoring/
17. Harris, M., Ghebreyesus, T., Tu, R.M., Kerkhove, V., Maria, D., Imogen, C.: Corinne: Covid-19 - world health organization (2020). https://www.who.int/docs/default-source/coronaviruse/transcripts/who-audio-emergencies-coronavirus-press-conference-full-20mar2020.pdf?sfvrsn=1eafbff_0
18. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge (2003)
19. Jocher, G., Stoken, A., Borovec, J.: ultralytics/yolov5: v3.1 (2020). https://doi.org/10.5281/zenodo.4154370
20. Kreiss, S., Bertoni, L., Alahi, A.: Pifpaf: Composite fields for human pose estimation. CoRR (2019). arXiv:1903.06593
21. Landing AI: Landing AI creates an AI tool to help customers monitor social distancing in the workplace (2020). https://landing.ai/landing-ai-creates-an-ai-tool-to-help-customers-monitor-social-distancing-in-the-workplace
22. Li, J., Wang, C., Zhu, H., Mao, Y., Fang, H.S., Lu, C.: Crowdpose: Efficient crowded scenes pose estimation and a new benchmark (2018). https://doi.org/10.48550/ARXIV.1812.00324
23. Lucas, B.D., Kanade, T.: An iterative image registration technique with an application to stereo vision. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'81, p. 674-679. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1981)
24. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS 2017 Workshop on Autodiff (2017). https://openreview.net/forum?id=BJJsrmfCZ
25. Sinha, P.K.: Camera Calibration. SPIE. https://doi.org/10.1117/3.858360.ch8
26. Khandelwal, P., Khandelwal, A., Agarwal, S., Thomas, D., Xavier, N., Raghuraman, A.: Using computer vision to enhance safety of workforce in manufacturing in a post covid world. arXiv:2005.05287 (2020)
27. Punn, N.S., Sonbhadra, S.K., Agarwal, S.: Monitoring covid-19 social distancing with person detection and tracking via fine-tuned yolo v3 and deepsort techniques. arXiv:2005.01385 (2020)
28. Qin, J., Xu, N.: Reaserch and implementation of social distancing monitoring technology based on ssd. Procedia Comput. Sci. **183**, 768–775 (2021). Proceedings of the 10th International Conference of Information and Communication Technology https://doi.org/10.1016/j.procs.2021.02.127 https://www.sciencedirect.com/science/article/pii/S1877050921006037
29. Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. CoRR (2015). arxiv:1506.02640
30. Shi, J., Tomasi: Good features to track. In: 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 593–600 (1994). https://doi.org/10.1109/CVPR.1994.323794
31. STEREOLABS: Using 3D cameras to monitor social distancing (2020). https://www.stereolabs.com/blog/using-3d-cameras-to-monitor-social-distancing/
32. Szabo, F.: The Linear Algebra Survival Guide: Illustrated with Mathematica. Academic Press Inc, Cambridge (2015)
33. WHO: Listings of WHO's response to COVID-19 (2020). https://www.who.int/news/item/29-06-2020-covidtimeline
34. Wojke, N., Bewley, A., Paulus, D.: Simple online and realtime tracking with a deep association metric. In: 2017 IEEE International Conference on Image Processing (ICIP), pp. 3645–3649. IEEE (2017). https://doi.org/10.1109/ICIP.2017.8296962
35. Yang, D., Yurtsever, E., Renganathan, V., Redmill, K.A., Ozguner, U.: A vision-based social distancing and critical density detection system for covid-19. arXiv:2007.03578 (2020)
36. Yang, Y.: FastMOT: High-Performance Multiple Object Tracking Based on YOLO, Deep SORT, and Optical Flow (2020). https://doi.org/10.5281/zenodo.4294717
37. Zhou, K., Yang, Y., Cavallaro, A., Xiang, T.: Omni-scale feature learning for person re-identification. arXiv:1905.00953 (2019)

Bilal Abdulrahman received his master's in Data Science from the CUNY Graduate Center, in 2022. He is, currently pursuing a PhD in computer science at the CUNY Graduate Center. He is a member of the City College Visual Computing Laboratory since 2020. His research interests include computer vision, and machine learning, currently focusing on human interaction and human body reconstruction.

Zhigang Zhu is current Herbert G. Kayser Chair Professor of Computer Science, at The City College of New York (CCNY) and The CUNY Graduate Center, where he directs the City College Visual Computing Laboratory (CcvcL). Dr. Zhu is an Associate Editor of the Machine Vision Applications Journal, Springer (2006 - now), and was Technical Editor, IEEE/ASME Transactions on Mechatronics (09/2010 - 09/2014). His research interests include 3D computer vision, multimodal sensing, human-computer interaction, virtual/augmented reality, and various applications in assistive technology, robotics, surveillance and transportation. He has published over 200 peer -reviewed technical papers in the related fields. His research has been supported by AFOSR, AFRL, ARO, DARPA, DHS, NSF, ODNI as well as industry. Dr. Zhu is a senior member of the IEEE, and a senior member of the ACM.