WILDTRACK: A Multi-camera HD Dataset for Dense Unscripted Pedestrian Detection – Supplementary Material –

Tatjana Chavdarova[1], Pierre Baqué[2], Stéphane Bouquet[2], Andrii Maksai[2], Cijo Jose[1], Timur Bagautdinov[2], Louis Lettry[3], Pascal Fua[2], Luc Van Gool[3], and François Fleuret[1]

[1]Machine Learning group, Idiap Research Institute & École Polytechnique Fédérale de Lausanne
[2]CVLab, École Polytechnique Fédérale de Lausanne
[3]Computer Vision Lab, ETH Zurich
{*firstname.lastname*}@{*idiap*[1],*epfl*[2],*vision.ee.ethz*[3]}*.ch*

Foremost, we elaborate in more detail the annotation procedure in Section 1. We then list details regarding the provided annotations in Section 2. Details on our implementation of the camera calibration are given in Section 3. Section 4 provides further details of the statistics summirized in Section 3.4 of the paper. In Section 5 we discuss recommended training and testing splits of the WILDTRACK dataset. Finally, we provide additional tracking results in Section 6.

1 Annotation process

Annotation tool. As an area of interest we consider a $12\times36m$ ground plane of the $3D$ space lying in the intersection of the fields of view of the seven cameras. We discretize this ground surface as a grid of 480×1440 points, what corresponds to an offset of $2.5cm$ in both directions. Given such a regular high-density grid of, at each location we construct a cylinder volume whose height and width correspond to the humans' average height and width. Each such cylinder projects into the separate 2D views as a rectangle. The position of these rectangles in all of the views is then calculated in pixel coordinates using the camera calibration. Finally, we use these pre-calculated projections to integrate them into our annotation tool.

The labelling tool is a Python-based web application. It is built with a very responsive design, and its graphical user interface (GUI) is illustrated in Fig. 1. Our annotation tool is hosted on a website[1], which was created and managed using Django. The source-code is also available for download[2].

For the selected frame to be annotated, the tool displays the seven corresponding images at the same time (see Fig. 1). In order to provide a multi-view annotation, the user of the tool first has to mark the placement of the bounding boxes. This is achieved by a *single click*, whose location should be at the feet of the person to be annotated, in either of the views where it is visible. Instantaneously, the boxes automatically appear in the views in which the person is visible. To complete the multi-view annotation, the user shall next adjust the position of the bounding boxes.

For this purpose, the keyboard arrows shall be used. More precisely, the *left*, *right*, *up* and *down* keys should be used in order to shift the $3D$ *imaginary* cylinder on the ground plane. To help annotators, the correspondence "key-direction" for each of the views is also depicted in the tool and optionally visible while annotating. In addition, a *zooming feature* can be used, that once a multi-view annotation is selected, allows for zooming-in the corresponding bounding boxes. This was implemented in order to make it easier for the annotators to obtain more precise locations of the annotations. The arrow key presses that translate into a movement of the $3D$ cylinder, are instantly visible in all of the views that capture the person currently being annotated. After getting used to the annotation process, annotators become more and more precise on the first step: placing the bounding boxes, which significantly reduces the time required to annotate as less adjustments are required.

[1]https://pedestriantag.epfl.ch/
[2]https://github.com/cvlab-epfl/multicam-gt

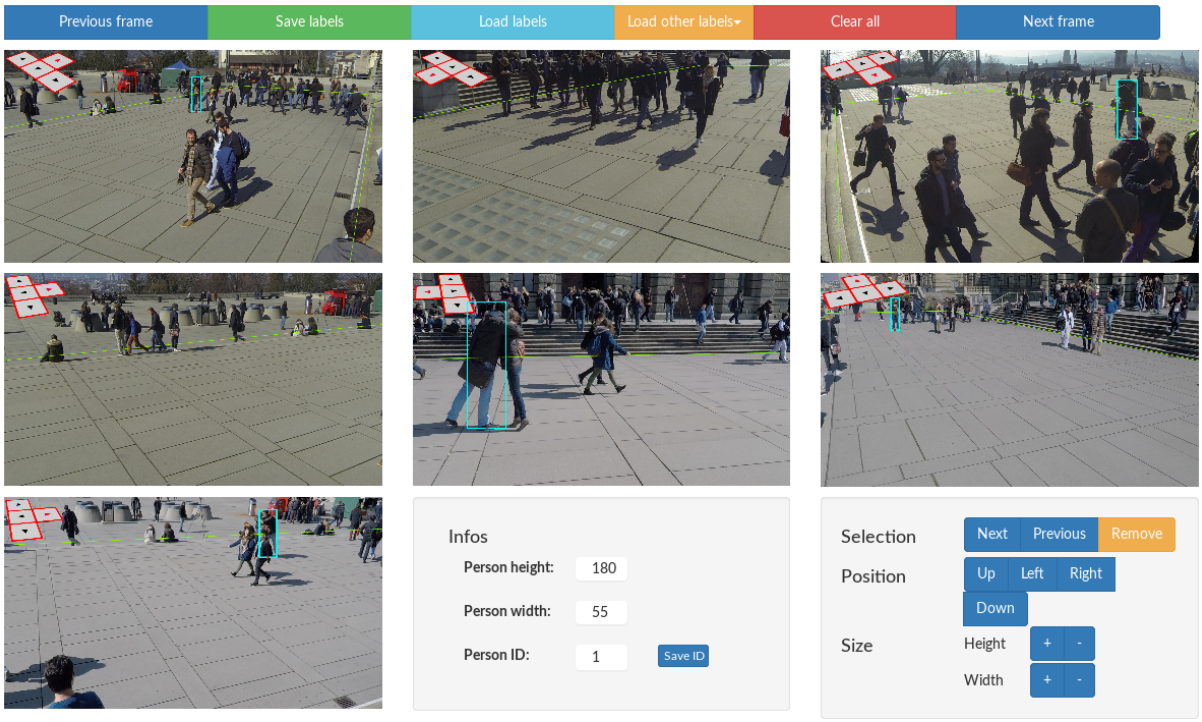

Figure 1: Graphical User Interface (GUI) of our multi-view labeller.

Once the frame has been fully labelled and the user has moved to the next frame, optionally (s)he is able to reload the annotations from the previous frame, traverse each of the annotations, and refine their positions. Additional features such as keyboard short-cuts are also supported for these utilities.

Finally, a more elaborate version of these instructions is provided in the annotation tool, accompanied with numerous illustrations.

Annotating on Mechanical Turk. We used Amazon Mechanical Turk [3] to obtain our annotations. Due to the risk of the annotators prioritizing profit over quality of the annotations, we were highly involved in the process.

In our experience of annotating frames of our dataset, pre-loading annotations from the previous frame, traversing these, and adjusting each, often proves less time-consuming then starting to annotate each multi-view frame from scratch. This motivated providing the feature of *pre-loading annotations* explained above. Hence, to help accelerate the annotation process the recruited annotators were assigned frames in batches of size 10. To ensure that this feature is not negatively utilised by the annotators, we also store flags indicating if these "imported" annotations have been adjusted or not.

As explained, annotators were found via Mechanical Turk. However, since the dataset is quite challenging, annotating locations in 3D for crowded scenes may require substantial attention and dedication. Despite all our efforts to make the tool easy to use, it turned out that most MT workers were reluctant to provide this level of effort and they were almost never achieving the required quality. We therefore had to select few workers to whom we personally explained the level of detail needed. They were then able to annotate with higher accuracy.

On average, annotating one frame takes $\sim$10 minutes for a trained annotator, and approximately half of that when importing the annotations from the previous frame.

2 Annotations

2.1 File formats

The annotations are provided in a separate file per each multi-view frame. Each annotation file is provided in the JavaScript Object Notation (JSON) open-standard file format. This format is human readable and programming language independent. Many programming languages integrate libraries that offer support for working with these files, including Python.

Each multi-view annotation contains the following information:

- **Person ID**: A unique identifier of a person appearing in the sequence.
- **3D location**: (X, Y) location of the target in meters on the ground plane with respect to the origin.
- **pixel coordinates in each of the views**: For each of the seven cameras, the detection location in pixel coordinates for that view are given: $\{(x^c_{min}, y^c_{min}, x^c_{max}, y^c_{max})\}$, $c = 1, \ldots, 7$.

2.2 Memory size

Images. We refer as a *frame* a set of 7 images, synchronized with the same time stamp. The extracted and pre-processed frames with removed distortions contain 36000×7 images, while each image is of size ∼2.9 MB. This corresponds to 10 frames per second for 1h and 7 cameras. Currently there are 400 annotated frames, at 2fps (see Section 3.4).

Videos. Each of the 7 videos is approximately 1:50h long, and of size ∼25GB.

3 Camera calibration

Intrinsic. The intrinsic calibration was obtained for each camera separately. For this purpose we used the *OpenCV* function *calibrateCamera* which provides also the distortion coefficients. Precisely, we used 3 radial distortion coefficients. In particular, we used the asymmetric circle grid provided by *OpenCV* with sizes of 4×11, and 20 frames to obtain each camera's intrinsic matrix. To obtain higher accuracy, we made sure that the target (the grid of circles), is captured in as many parts of the field of view of the camera as possible.

Extrinsic. In our implementation, for each of the seven views we used 23, 26, 15, 19, 21, 28 and 19 pairs of points, respectively. We used the *OpenCV*'s module *solvePnP* [2], which given the intrinsics provides the rotation and the translation vector. The 3D measurements and the annotated corresponding points will also be made available, so as camera calibration methods could make use of these.

Bundle adjustment. In our implementation, we used the open source C++ library *Ceres* [1], which offers extensive support for bundle adjustment problems. We used linear optimisation which in Ceres is referred to as *Iterative Schur*.

4 Additional statistics

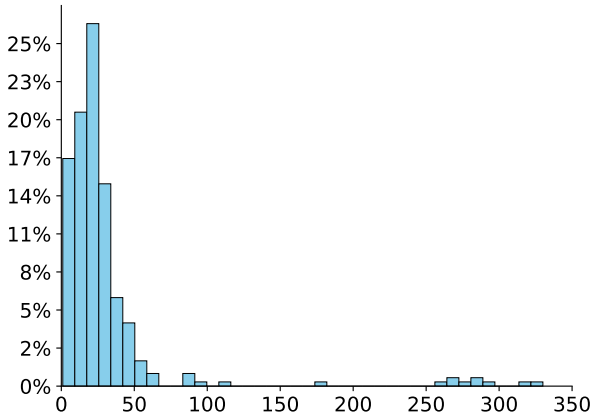

Figure 2: Histogram of the number of frames in which one person appears: the normalized number of different identites (y-axis) that appear within a range of number of frames (x-axis).

Fig. 2 depicts the number frames in which a person appears. In particular, we consider a frame rate of 2 fps, a total number of frames of 400, and 313 different identities. On average, each person appears in 30.41(47.87) frames, and the mode is 22 frames.

5 Recommended splits of the WILDTRACK dataset

We regard two use-cases of the WILDTRACK dataset, and we discuss recommended partitions for each. Please consider visiting the website for downloading the dataset[3], for up to date details.

Scenario A: Supervised methods. We recommend that the last 10% of the annotated frames at 2 fps are used for testing. This amounts to a total of 40 frames at 2 fps. For training one shall use the remaining portion of the dataset, with optional sampling frame rate.

Scenario B: Unsupervised methods. In this case, we recommend that the entire annotated portion at a fixed frame rate of 2 fps is used for benchmarking unsupervised methods. The remaining portion for which annotations are not provided can be used for training, using an optional sampling rate.

6 Additional tracking benchmarks

Table 1 shows additional tracking results, where we use the same notation for the methods as in the paper (see Section 4.2 in the paper).

Table 1: Additional tracking results on the WILDTRACK dataset.

Method	IDF1	IDP	IDR	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP
ResNet-DeepMCD+KSP	62.5	84.9	49.5	11	11	19	154	2081	35	30	50.9	75.1
ResNet-DeepMCD+KSP+ptrack	64.2	93.1	49.0	10	9	22	49	2239	5	5	50.4	75.9
ResNet-View 1+KSP	28.5	18.7	60.7	34	4	0	10050	257	162	58	-140.9	59.0
ResNet-View 1+KSP+ptrack	30.2	20.1	60.6	30	8	0	9173	410	131	51	-123.5	58.0
ResNet-View 2+KSP	29.1	19.4	58.8	28	6	1	8428	265	172	47	-121.3	50.7
ResNet-View 2+KSP+ptrack	31.4	21.2	60.6	28	6	1	7698	249	128	31	-101.6	50.2
ResNet-View 3+KSP	25.8	17.0	53.7	35	4	1	9874	286	177	52	-133.5	51.2
ResNet-View 3+KSP+ptrack	27.2	18.1	54.2	33	5	2	9208	402	150	46	-120.5	49.1
ResNet-View 4+KSP	20.5	12.6	54.4	14	5	1	4362	137	42	16	-255.3	60.5
ResNet-View 4+KSP+ptrack	22.1	13.9	54.4	13	2	5	3904	180	32	11	-222.1	60.3
ResNet-View 5+KSP	39.7	32.6	50.9	20	13	3	2560	598	117	79	5.8	54.2
ResNet-View 5+KSP+ptrack	41.7	35.0	51.7	18	12	6	2334	672	94	54	10.9	55.2
ResNet-View 6+KSP	26.6	17.5	55.4	34	4	1	10200	375	172	77	-136.1	52.2
ResNet-View 6+KSP+ptrack	29.4	19.9	56.4	30	8	1	8860	498	127	51	-108.4	52.8
ResNet-View 7+KSP	38.6	27.1	67.0	22	3	0	4488	171	72	28	-61.1	65.1
ResNet-View 7+KSP+ptrack	41.7	30.3	66.8	19	3	3	3791	253	49	21	-39.4	64.9

References

[1] S. Agarwal, K. Mierle, and Others. Ceres solver. http://ceres-solver.org.

[2] G. Bradski. Opencv. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] M. Buhrmester, T. Kwang, and S. D. Gosling. Amazon's mechanical turk: A new source of inexpensive, yet high-quality, data? *Perspectives on Psychological Science*, 6(1):3–5, 2011. PMID: 26162106.

[3] https://cvlab.epfl.ch/data/wildtrack