Video Surveillance for Road Traffic Monitoring

María Gil Oscar Mañas Laura Mora Roger Casals
Universitat Politècnica de Catalunya
{maria.gilaragones oscmansan lmoraballestar rogercasalsvilardell}@gmail.com

Abstract

This paper presents a comparison between different multi-object single and multi-camera tracking techniques implemented by the authors and accepted to the 7th Workshop on Traffic Road Monitoring hosted by Universitat Politècnica de Catalunya in the Spring of 2020. It is based on a particular task of the AI City Challenge 2020, which proposes to track vehicles across multiple cameras both at a single intersection and across multiple intersections spread out across a city. The performance of the implemented techniques is measured using standard metrics for video analysis such as mAP and IDF1. The code is available at `https://github.com/mcv-m6-video/mcv-m6-2020-team2`.

1. Motivation

The main motivation of this project came from the aim to solve the CVPR 2020 AI City Challenge. This challenge arises from the need to make transportation systems smarter.

Among the different tasks proposed in the challenge, this project focuses on solving Track 3: City-Scale Multi-Camera Vehicle Tracking. The goal of this task is to detect and track targets across multiple cameras both at a single intersection and across multiple intersections spread out across a city. The provided dataset contains 3.58 hours of videos collected from 46 cameras spanning 16 intersections in a mid-sized US city.

2. Related work

Image-based ReID The state-of-the-art in ReID uses metric learning with different loss functions, such as hard triplet loss [11, 3], center loss [14], and contrastive loss [1]. More recently, [4] explores the advances in batch-based sampling for triplet embedding that are used in person ReID solutions and applies them to the vehicle ReID problem. They compared different sampling variants, such as batch all (BA), batch hard (BH), batch sample (BS), and batch weighted (BW), adopted from [3, 10].

Object detection and single-camera tracking The tracking-by-detection paradigm is a state-of-the-art tracking method, consisting in detecting objects in each frame and then relating those objects to form tracks. In our experiments, we use the provided baseline detections from well-know detection methods such as YOLOv3 [9], SSD512 [6] and Mask R-CNN [2]. State-of-the-art MTSC methods include DeepSORT [15], which combines deep learning features with Kalman-filter-based tracking and the Hungarian algorithm for data association, TC [13], which applies tracklet clustering through optimizing a weighted combination of cost functions, and MOANA [12], which employs similar schemes for spatio-temporal data association, but using an adaptive appearance model.

Spatio-temporal analysis For the multi-camera tracking problem, we draw inspiration from several multi-camera vehicle re-identification methods. PROVID [8] proposes a progressive and multimodal vehicle ReID framework, in which a spatio-temporal-based re-ranking scheme is employed. 2WGMMF [5] is a method which learns the transition time between camera views by using two-way Gaussian mixture model features. In FVS [13], however, since no training data is provided, the temporal distribution is predefined based on the estimated distance between cameras. In our work, we also use the estimated GPS location of cameras.

3. Method

To achieve the goal of this project, the work has been divided into two stages: a first stage limited to a single camera and a second stage in which the knowledge acquired will be applied to extend from one to multiple cameras.

3.1. Multi-Target Single-Camera tracking

3.1.1 Object detection

The basic procedure to conduct the tracking consists on an object detection step followed by the tracking technique that processes the detections. However, as the AI City Challenge 2020 already provides files containing the detections made

by different state of the art detectors, such as Mask-RCNN, SSD512 and YOLO3, it has not been necessary to develop an object detection system.

3.1.2 Tracking techniques

Maximum overlap The maximum overlap technique matches the detected bounding boxes in consecutive frames using IoU.

Each bounding box on frame N + 1 is exclusively assigned to the track of the closest (highest IoU) bounding box on frame N. The track of those bounding boxes on frame N that do not intersect with any bounding box on frame N + 1 are terminated. The remaining bounding boxes which are not assigned to any track are the starting point of a new track.

As the technique goes, the bounding boxes at frame N+1 are not modified. However, we have introduced a refinement method which improves the quality of the detections: given the tracked detections of a given object, we predict its future position with interpolation; match object boxes detected in frame N+1; finally, refine matched box by computing the center between the predicted position and the detected one. This refinement should adjust misaligned detections between frames, and provide smoother tracking. It can also compensate for temporarily losing objects.

Kalman filter The Kalman filter is a two-step procedure which allows to track objects across frames. It consists on a prediction that estimates the future position of an object followed by an update after a new observation arrives, which refines the future position by using the observed data. It works as the optimal estimation for the state of a system under the assumptions that the system behaves as a linear model with Gaussian noise. However, given that we are tracking moving vehicles which do not necessarily follow these assumptions, the tracking with a Kalman filter is not optimal in this case.

Optical flow In order to improve the object tracking by maximum overlap, we add the information obtained from optical flow, which allows to estimate the apparent motion of each pixel in the image.

In the original tracking method, we compute the intersection over union to find the detection with maximum overlap. We improve over this by displacing the detected bounding box from the previous frame according to the estimated motion from the optical flow.

3.2. Multi-Target Multi-Camera tracking

3.2.1 Visual representation of tracks

To build a representation of tracks we have used a metric learning approach: image embeddings of the same identity should be "close" to each other, while different identities should be "far away". In this subsection we are going to first detail how we built the dataset and then describe how we trained the encoder to extract meaningful representations of tracks.

Dataset Track 3 of the AI City Challenge 2020 is composed by three train sequences —S01, S03 and S04—, with several viewpoints each. From this data, we have selected S01 and S04 as our training set and S03 is used for validation (see Table 1 for more information on the dataset).

To train the encoder we need to separate each identity. To this end, we process the camera recordings for each of the sequences together with the provided ground truth file to extract the image crops for each car. Crops belonging to the same identity are stored in a file structure such as *sequence/identity/camera-crop.jpg*.

Table 1. Dataset

Measure	Train	Validation
Number classes	166	18
Total images	38074	6174
Mean images/class	230	343

Online Mining Triplet Loss We follow the approach proposed in [11], which generates triplets on-the-fly for each input batch during training. Thus, it is important that the batch has a balanced representation of samples per identity in order to obtain good triplets. A batch of B instances is created by selecting P identities with K samples each.

To generate the triplets, we follow a *batch-hard* strategy. For each anchor, we select the hardest positive (maximum distance among samples from the same identity) and the hardest negative (minimum distance among samples from a different identity), which generates $B = P \cdot K$ triplets per batch.

Training details We use a ResNet18 backbone pretrained on ImageNet, with an Adam optimizer with a constant learning rate of 0.001 and a weight decay of 0.0001, an embedding size $D = 128$ dimensions, $K = 10$ samples/identity, $P = 100$ identities/batch and a multi-camera aware batch sampler. In addition, we introduced extensive data augmentation, including random horizontal flips and color jittering, to gain some degree of viewpoint/pose invariance. Figure 1 shows a t-SNE representation of the embedding obtained with these parameters.

During our training experiments, we have observed that P, K and the embedding size D are the parameters that af-

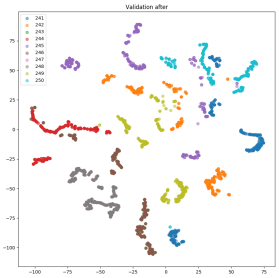

Figure 1. T-SNE embedding representation for the validation sequence S03.

fect model performance the most. A low embedding size D can lead the network to overfit. Using a high number of samples/identity K improves the embedding performance, since it is easier to find harder positives (e.g. car images belonging to the same identity but captured from different cameras). Also, using high values of P helps to create harder negatives for the network to learn from.

3.2.2 Spatio-temporal analysis

We leverage the spatial and temporal information provided by the AI City Challenge data to reduce the amount of comparisons between track representations to group them by identity. This makes the re-identification method less dependent on a good representation of tracks, and reduces the amount of false positives when matching tracks from different cameras.

The reasoning behind this spatio-temporal filtering is based on two observations/assumptions. First, there is no spatial intersection between cameras, since all of them are pointing to different road sections. As a consequence, cars appear in a certain order in the cameras.

With this in mind, given a track from a camera, we reduce the search space of possible tracks belonging to the same identity to other cameras that appear roughly in the same direction where the car is going and, from those cameras, other tracks that have a later timestamp (and a similar direction).

Spatial analysis Given the homography between the image plane and the ground plane for each camera, we can back-project car detections in image coordinates to obtain the GPS position (latitude and longitude) of the car in a frame. By subtracting the GPS position between a later and an earlier frame, we obtain a rough estimate of the car direction in world coordinates. In addition, we are provided with an image showing the camera locations. Using also the point of view of the videos, we search the approximate GPS position of each camera in Google Maps, similarly to [8]. By subtracting the GPS position between the camera and the car we can obtain the direction of the camera with respect to the car. Finally, we compute the angle between the car trajectory and the camera. If that angle is less than 45^{o} we consider that the car is moving towards the camera.

Temporal analysis We are provided with the initial timestamp of each video sequence, which we can use for synchronization. Since we also know the framerate of each clip, we can compute the timestamp of a given video frame as follows:

$$t_{frame} = t_{start} + \frac{n_{frame}}{fps} \tag{1}$$

Thus, since we know which frames comprise a track, we can compute its initial and final timestamps. With this information, together with the cameras that a car could visit next, we can limit the search space to those tracks in upcoming cameras that happen after the current track.

3.2.3 Vehicle re-identification

Once we have a global representation for each track, we need to define how to match them and decide which ones are actually following the same vehicles across multiple cameras. A first approach is based on the exhaustive comparison of every track obtained from MTSC, using a clustering method to find groups of tracks belonging to the same identity; we refer to it as exhaustive re-identification. However, we can leverage spatio-temporal coherence to filter the number of comparisons between tracks (see section 3.2.2). We propose two different methods to re-identify cars after this filtering: sequential and graph-based.

Sequential re-identification For every track in every camera we compare its representation against every filtered candidate in other cameras. Matches are defined as the candidates with the closest embedding to the query, up to a certain threshold. Matches with a distance higher than the threshold are disregarded to avoid false positives.

Sequential matching considers that a track can be matched at most with two other tracks: one from the previous camera the car has left and another from the next camera where it could appear. When two tracks are matched, the connection defines the appearance order of the car in them. Limiting the matches between cars across the multiple comparisons, we end with a set of tracks connected to one, two or no tracks.

If we define starting tracks as the ones that do not have connections in previous tracks, we can propagate their id's across their connected tracks until there is no consecutive track. Moreover, tracks that are not connected to any other

are filtered as they identify cars that should only appear in one camera.

We are aware this method is sensitive to the order of the cameras in which we compare the tracks, and it also requires that a car does not change its direction in between cameras.

Graph-based re-identification The shortcomings of the previous method mainly come from constraining the matching of each track to a sequential process. In order to overcome these flaws, we devise a more general method based on formulating the track matching problem as a graph problem.

To build the graph, we consider the individual tracks of all cameras as the vertices, and we add an edge between each pair of vertices whose associated tracks are similar based on the previously trained similarity metric (see section 3.2.1).

The set of tracks corresponding to the same identity across cameras will form a clique[1] in the graph [7]. We iteratively select the maximal clique in the graph, mark all its nodes as belonging to the same identity, and remove all those nodes and their incident edges from the graph; this avoids that a track is assigned to more than one identity.

To manage the graph structure and find cliques, we used the Python package *NetworkX*[2].

4. Evaluation

To evaluate the performance of the implemented methods, we use standard metrics for video analysis, which are available in the *py-motmetrics* Python package[3]: Identification F1 score (IDF1), Identification Precision (IDP) and Identification Recall (IDR).

4.1. Multi-Target Single-Camera tracking

In order to evaluate the single-camera tracking methods, it has been necessary to adapt them to the ground truth, as it does not consider small detections nor those tracks corresponding to parked cars. We discard small bounding boxes and ephemeral tracks (those with a low number of detections). Moreover, tracks corresponding to static cars are removed, considering the maximum of the euclidean distances between all pairs of centroids.

Average IDF1 results across cameras of sequences S01, S03 and S04 are presented in table 2. The complete results for all cameras are shown in appendix A. Even though our method outperforms the baseline for sequence S03, we must take into account that we adjusted the hyperparameters with that sequence. Our method performs worse for test sequences S01 and S04. Moreover, no detector works clearly better than the rest. For sequences S03 and S04 Yolo3 works better but for sequence S01 the best performance is achieved by SSD512. Although the Kalman filter method produces better IDF1 than maximum overlap on some cameras of the sequences, its performance is overall worse.

Finally, we can see in the IDF1 results for the three sequences that using optical flow does not provide any improvement at all. We believe this is because the bounding boxes are already good enough.

4.2. Multi-Target Multi-Camera tracking

Table 3 shows the Multi-Track Multi-Camera results, using the three different re-identification methods previously discussed. As expected, exhaustive re-id yields the worst IDF1 score. It has the best recall and the worst precision among all methods, indicating that there are excessive detections. Sequential ReID has been evaluated using both Euclidean and cosine distances, setting the similarity threshold to 1.5 in both cases. Lowering the threshold slightly decreases performance while higher values maintain the results. Despite the mentioned shortcomings of this method, graph-based ReID yields only slightly better IDF1 results. We argue that all these re-identification methods are constrained by the quality of the visual representation of tracks.

Qualitative results show that we are able to track cars between two cameras but our system loses precision when extending those tracks to more cameras.

5. Conclusions

We have implemented a pipeline able to re-identify cars between two cameras, with decreasing performance when extending to more views. Our method leverages spatial and temporal information, but it is specifically tailored for the case where cameras are located at different intersections, such as in sequence S03. Thus, the system may not work as well in other scenarios. Although the learned similarity metric for track representation does not work well for car identities appearing in multiple cameras, we have been able to compensate it by adding these spatio-temporal constraints.

As future work, we should improve the track representation. To that end, we could train the metric learning model from scratch, since the pretrained weights may be limiting the new features the model is able to learn. Finally, we could improve the graph-based ReID method by using track similarities as weights in the graph representation, and then selecting cliques with maximum cumulative similarity instead of maximum number of vertices.

[1] A clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent; that is, its induced subgraph is complete.

[2] `https://networkx.github.io/`

[3] `https://github.com/cheind/py-motmetrics`

Table 2. Average IDF1 results for our best method and a baseline, across the different cameras of sequences S01, S03 and S04.

Sequence	S1	S3	S4	Average
Yolo3, overlap (ours)	66.35%	**63.62%**	63.89%	**64.62%**
Yolo3, TC (baseline)	**69.65%**	56.06%	**65.52%**	63.74%

Table 3. Multi-target multi-camera results for every ReID method on Sequence 3.

ReID method	IDF1	IDP	Precision	Recall
Exhaustive	0.1556	26.58%	63.54%	**98.85%**
Sequential (Euclidean)	0.4456	45.66%	81.42%	77.57%
Sequential (Cosine)	0.4520	46.20%	79.29%	75.91%
Graph-based	**0.4679**	**49.98%**	**82.71%**	72.80%

References

[1] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[2] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B Girshick. Mask r-cnn. corr abs/1703.06870 (2017). *arXiv preprint arXiv:1703.06870*, 2017.

[3] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[4] Ratnesh Kuma, Edwin Weill, Farzin Aghdasi, and Parthasarathy Sriram. Vehicle re-identification: an efficient baseline using triplet embedding. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2019.

[5] Young-Gun Lee, Zheng Tang, and Jenq-Neng Hwang. Online-learning-based human tracking across non-overlapping cameras. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2870–2883, 2017.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[7] Wenqian Liu, Octavia Camps, and Mario Sznaier. Multi-camera multi-object tracking. *arXiv preprint arXiv:1709.07065*, 2017.

[8] Xinchen Liu, Wu Liu, Tao Mei, and Huadong Ma. Provid: Progressive and multimodal vehicle reidentification for large-scale urban surveillance. *IEEE Transactions on Multimedia*, 20(3):645–658, 2017.

[9] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[10] Ergys Ristani and Carlo Tomasi. Features for multi-target multi-camera tracking and re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6036–6046, 2018.

[11] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[12] Zheng Tang and Jenq-Neng Hwang. Moana: An online learned adaptive appearance model for robust multiple object tracking in 3d. *IEEE Access*, 7:31934–31945, 2019.

[13] Zheng Tang, Gaoang Wang, Hao Xiao, Aotian Zheng, and Jenq-Neng Hwang. Single-camera and inter-camera vehicle tracking and 3d speed estimation based on fusion of visual and semantic features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 108–115, 2018.

[14] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *European conference on computer vision*, pages 499–515. Springer, 2016.

[15] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.

Appendices

A. Multi-Target Single-Camera tracking results

Table 4. IDF1 results on the different cameras of sequence 3.

Camera	c10	c11	c12	c13	c14	c15	Average
Mask-RCNN, overlap	89.70%	53.41%	52.08%	75.86%	86.04%	21.52%	63.10%
SSD-512, overlap	86.31%	40.33%	54.59%	78.84%	84.86%	0.00%	57.49%
Yolo3, overlap	93.33%	40.46%	**55.96%**	83.27%	83.68%	**25.00%**	**63.62%**
Yolo3, overlap + optical flow	93.33%	40.46%	**55.96%**	83.27%	83.68%	**25.00%**	**63.62%**
Mask-RCNN, Kalman filter	87.08%	**53.54%**	54.98%	67.37%	67.58%	21.62%	58.69%
SSD-512, Kalman filter	79.11%	25.20%	55.00%	65.75%	68.73%	0.00%	48.96%
Yolo3, Kalman filter	**95.01%**	27.54%	53.06%	72.06%	**87.68%**	23.44%	59.80%
Baseline: Yolo3, TC	81.98%	49.55%	17.41%	**87.07%**	79.38%	20.99%	56.06%

Table 5. IDF1 results on the different cameras of sequence 1.

Camera	c01	c02	c03	c04	c05	Average
Mask-RCNN, overlap	63.09%	68.50%	76.59%	77.85%	**41.81%**	65.57%
SSD-512, overlap	67.02%	74.65%	78.75%	**82.75%**	38.35%	68.30%
Yolo3, overlap	62.71%	**84.46%**	**83.45%**	70.41%	30.73%	66.35%
Yolo3, overlap + optical flow	62.71%	**84.46%**	**83.45%**	70.41%	30.73%	66.35%
Mask-RCNN, Kalman filter	58.62%	69.47%	71.66%	65.39%	34.20%	59.87%
SSD-512, Kalman filter	57.67%	73.92%	68.99%	55.29%	30.40%	57.25%
Yolo3, Kalman filter	59.43%	77.54%	77.00%	65.00%	25.36%	60.87%
Baseline: Yolo3, TC	**78.43%**	77.06%	73.95%	78.50%	40.33%	**69.65%**

Table 6. IDF1 results on the different cameras of sequence 4 (i).

Camera	c16	c17	c18	c19	c20	c21	c22
Mask-RCNN, overlap	67.93%	**56.31%**	47.74%	87.72%	53.02%	56.29%	71.57%
SSD-512, overlap	73.89%	51.37%	64.79%	82.31%	53.24%	65.27%	68.54%
Yolo3, overlap	80.07%	48.15%	62.26%	87.44%	50.41%	93.13%	66.83%
Yolo3, overlap + optical flow	80.07%	48.15%	62.26%	87.44%	50.41%	93.13%	66.83%
Mask-RCNN, Kalman filter	52.11%	45.12%	45.87%	7.61%	**59.30%**	70.08%	71.52%
SSD-512, Kalman filter	57.00%	53.15%	56.50%	6.14%	55.98%	70.27%	30.11%
Yolo3, Kalman filter	82.15%	44.14%	54.47%	88.24%	52.58%	**94.43%**	42.47%
Baseline: Yolo3, TC	**85.04%**	51.69%	**74.75%**	**91.79%**	45.78%	94.19%	**73.01%**

Table 7. IDF1 results on the different cameras of sequence 4 (ii).

Camera	c23	c24	c25	c26	c27	c28	c29
Mask-RCNN, overlap	**62.82%**	**61.68%**	62.13%	83.69%	56.69%	54.83%	**59.65%**
SSD-512, overlap	53.29%	53.49%	35.08%	84.14%	60.14%	55.33%	38.99%
Yolo3, overlap	50.23%	52.63%	58.64%	81.37%	64.67%	**61.93%**	54.64%
Yolo3, overlap + optical flow	50.23%	52.63%	58.64%	81.37%	64.67%	**61.93%**	54.64%
Mask-RCNN, Kalman filter	41.70%	50.00%	**66.90%**	80.61%	60.54%	45.88%	53.09%
SSD-512, Kalman filter	29.45%	38.46%	40.76%	66.37%	55.04%	57.00%	40.75%
Yolo3, Kalman filter	37.44%	52.31%	59.59%	83.72%	53.35%	59.52%	54.41%
Baseline: Yolo3, TC	36.00%	29.03%	55.89%	**85.09%**	**83.96%**	58.94%	56.07%

Table 8. IDF1 results on the different cameras of sequence 4 (iii).

Camera	c30	c31	c32	c33	c34	c35	c36
Mask-RCNN, overlap	67.43%	12.78%	**76.66%**	78.03%	69.07%	78.93%	64.66%
SSD-512, overlap	69.51%	12.96%	58.18%	**80.41%**	68.11%	72.06%	**70.02%**
Yolo3, overlap	63.24%	24.50%	59.16%	78.40%	66.95%	75.02%	64.92%
Yolo3, overlap + optical flow	63.24%	24.50%	59.16%	78.40%	66.95%	75.02%	64.92%
Mask-RCNN, Kalman filter	48.33%	15.79%	47.95%	76.29%	53.74%	72.94%	59.51%
SSD-512, Kalman filter	51.00%	17.28%	46.88%	64.69%	60.71%	72.51%	56.39%
Yolo3, Kalman filter	52.53%	22.18%	43.03%	69.80%	50.38%	69.84%	64.71%
Baseline: Yolo3, TC	**86.38%**	**64.90%**	61.46%	69.26%	**70.72%**	**79.27%**	67.82%

Table 9. IDF1 results on the different cameras of sequence 4 (iv).

Camera	c37	c38	c39	c40	Average
Mask-RCNN, overlap	66.05%	65.59%	71.68%	57.24%	63.61%
SSD-512, overlap	**66.72%**	71.20%	62.48%	**73.37%**	61.80%
Yolo3, overlap	53.12%	**73.32%**	69.95%	56.19%	63.89%
Yolo3, overlap + optical flow	53.12%	**73.32%**	69.95%	56.19%	63.89%
Mask-RCNN, Kalman filter	46.06%	62.21%	67.78%	54.03%	54.19%
SSD-512, Kalman filter	48.86%	66.43%	72.17%	40.12%	50.16%
Yolo3, Kalman filter	37.50%	60.99%	**81.92%**	27.83%	57.58%
Baseline: Yolo3, TC	31.39%	61.19%	74.23%	50.13%	**65.52%**