Integrating and Testing SOTA Multi-Object Tracking Algorithms with the YOLOv8 Detector

Nicklas Vraa, AU592379 Mathias Vraa, AU618687

***Abstract*—In this paper, we attempt to integrate discrete tracking algorithms on top of the state-of-the-art for object detection, namely the YOLOv8 object detector. We also analyze and discuss how tracking is traditionally approached as well as emerging and promising alternatives. Accompanying this paper is our GitHub repository ← (clickable).**

***Index Terms*—Computer Vision, Online Multi-Object Tracking, YOLO, DeepSORT, StrongSORT, BYTE, ByteTrack, BoT-SORT, OC-SORT, Deep-OC-SORT, FairMOT, MOTA, HOTA.**

1. INTRODUCTION

Online multi-object tracking (MOT) is arguably the area of computer vision with the most practical applications, e.g. autonomous driving, robot navigation and video surveillance. It is important to distinguish between *detection* and *tracking*. Tracking involves detection but also *re-identification* i.e., assigning unique ID's to objects and keeping track of them spatially and temporally, e.g. throughout the duration of a video clip. It is certainly possible to mimic tracking by simply detecting and drawing and bounding boxes around objects for each new frame in a video. However, *detection* alone does not distinguish between multiple objects of the same class appearing in the same scene, or objects being temporarily occluded or blurred. With detection alone, an object that cannot be seen, cannot be tracked. Humans easily track multiple objects which are both identical and occluded, so why shouldn't a computer be able to do the same? In this paper, we will be exploring and comparing SOTA (state-of-the-art) approaches to online multi-object tracking, i.e. for video being provided frame-by-frame. Currently, tracking-by-detection has become the most effective paradigm for MOT [1], which is why we will be looking specifically at three SOTA tracking algorithms that can be added on top of a high-performance detector, e.g. YOLOv8. These are StrongSORT [2], BoT-SORT [3] and Deep OC-SORT [4], which all are iterations on previous high-performing algorithms.

2. THEORY

Before analyzing the tracking algorithms, we will describe the common theory on top of which these are built. Firstly, we describe how we measure performance for object trackers, as well as critical theories that are included in most tracking algorithms. Finally, we discuss the YOLOv8 detector.

2.1. MEASURING TRACKING PERFORMANCE

When evaluating a MOT model, we need measures for accuracy and precision, which take into account re-identification performance. For this, there are five units-of-measure frequently used in the literature: MOTA, MOTP, IDF1, IDSw and HOTA.

MOTA is a measure of **m**ultiple **o**bject **t**racking **a**ccuracy, which combines the number of false positives fp_t, missed targets m_t and identity mismatches mme_t [5]. Ground truth is g_t. MOTA is defined as:

$$\text{MOTA} = 1 - \frac{\sum_t (m_t + \text{fp}_t + \text{mme}_t)}{\sum_t g_t} \tag{1}$$

MOTP is a measure of **m**ultiple **o**bject **t**racking **p**recision and similar to the classic mAP metric, where c_t is the number of matches found at time t, and for each of these matches, d_t^i is the distance between the object o_i and its corresponding hypothesis. It is defined as:

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{2}$$

IDF1 is the ratio of correctly identified detections over the average of ground truth and predicted detections [6], i.e.:

$$\text{IDF}_1 = \frac{2\text{IDTP}}{2\text{IDTP} + \text{IDFP} + \text{IDFN}} \tag{3}$$

where TP: true positives, FP: false positives and FN: false negatives - all of which only apply to the identification. **IDSw** is simply the total number of id-switches throughout.

HOTA (**h**igher **o**rder **t**racking **a**ccuracy) is another important and more recent measure for MOT performance, which explicitly balances the effect of performing accurate detection, association and localization into a single unified metric [7]. It also aligns better with human perception of performance, and breaks into sub-metrics, each handling a specific error type. These may simply be thought of as three intersection-over-union (IoU) measures:

- **LocA**, which measures the spatial alignment between one predicted detection and one ground-truth detection. The overall localization accuracy is the average over all pairs of matching predicted and ground-truth detections IoU in the dataset:
$$\text{LocA} = \frac{1}{|\text{TP}|} \sum_{c \in \text{TP}} \text{Loc-IoU(c)} \tag{4}$$
- **DetA**, which measures the alignment between the set of all predicted detections and the set of all ground-truth detections. The overall detection accuracy is simply the detection IoU over the entire dataset. Notice that this is simply the definition of IoU.
$$\text{DetA} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FN}| + |\text{FP}|} \tag{5}$$
- **AssA**, which measures how well a tracker links detections over time into the same identities, given the ground-truth set of identity links in the ground-truth tracks.

$$\text{AssA} = \frac{1}{|\text{TP}|} \sum_{c \in \text{TP}} \text{Ass-IoU}(c) \tag{6}$$

One predicted detection may overlap with more than one ground-truth detection (and vice versa). The Hungarian algorithm is used to determine a one-to-one matching between predicted and ground-truth detections for both DetA and AssA, based on a Loc-IoU threshold, which is also denoted as α.

HOTA is the geometric mean of the detection score and the association score, across each possible α. By integrating with respect to α, the localization accuracy is included in the final metric:

$$\text{HOTA} = \int_{0<\alpha\leq 1} \sqrt{\text{DetA}_\alpha \cdot \text{AssA}_\alpha} \tag{7}$$

HOTA not only measures how good a tracker is, but also *what* it is good at and where it may be lacking, which is crucial for understanding the underlying behavior of a tracker. We will be conducting all of the above measures on our chosen tracking algorithms.

2.2. MAHALANOBIS DISTANCE

Many trackers need a way to measure the distances between a point-vector and a distribution. If we simply measured the euclidian distance between the point and the center of the distribution, points P_1 and P_2 in fig. 1 would be of equal distance to the distribution, which is naturally not the case.

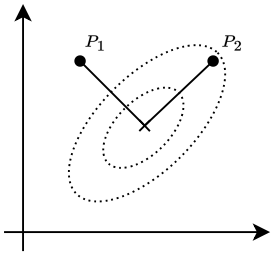


Figure 1: Illustration of why euclidean distance does not work for distributions.

Mahalanobis' distance is a multidimensional distance metric and is defined as:

$$D_M = \sqrt{(x-\mu)^T \cdot C^{-1} \cdot (x-\mu)} \tag{8}$$

Where μ is a vector of the mean values of each independent variable and C is the covariance matrix.

The term $(x - \mu)$ is simply the distance between the point and mean. Dividing by (or multiplying by the inverse of) the covariance matrix is just a generalization of standardization, i.e. $(x - \mu)/\sigma$. If dimensions are strongly correlated, then the covariance will be high, which would reduce the overall distance measure. Mahalanobis addresses both the problems of scale and correlation of variables [8].

2.3. THE KALMAN FILTER

As all of our chosen approaches rely on the Kalman filter, we will briefly review this and how it applies to object tracking. The Kalman filter is an algorithm that looks at multiple frames over time and estimates the future position of objects. This way, the motion of an object can be modeled, improving the tracking accuracy of objects, even when they are occluded.

The Kalman filter is a whole subject deserving of its own book, so only a brief overview of the concept is given here. The Kalman filter uses a dynamic model to estimate the outcome of a system based on the state of the system. For object tracking, the system is a moving object, so we use the laws of motion as a dynamic model and take into account states like velocity and acceleration. A dynamic model could be Newton's law of motion that calculates the expected position x based on the following states: initial position x_0, initial, initial velocity v_0, acceleration a and the time interval Δt.

$$x = x_0 + v_0 \Delta t + \frac{1}{2} a \Delta t^2 \tag{9}$$

The Kalman filter uses multiple sequential measurements across multiple frames to determine the mean and variance of the expected position and maps them onto the Gaussian distribution. This accounts for noise caused by measurement inaccuracy or environmental processes acting on the object, making it behave non-ideally.

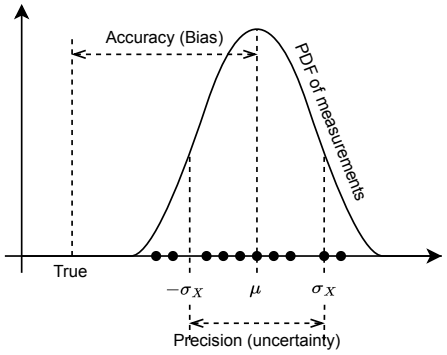


Figure 2: Illustration showing the calculated values mapped onto the Gaussian distribution.

2.4. THE HUNGARIAN ALGORITHM

As all of our chosen approaches rely on the Hungarian algorithm, we will briefly review it and how it applies to object tracking. The Hungarian method is an algorithm that solves the optimal assignment problem [9]. It is an optimization algorithm that is able to find the optimal trace through a matrix. The gist of the algorithm and it's relation to object tracking is best explained with a simple example. Say we are tracking bounding boxes and detect 3 objects and track 3 objects (fig. 3). Using the IOU as a loss function, we want to find the optimal combination that yields the highest IOU value. As there are only 6 combination ($!3 = 6$), this can easily be brute forced, and we find the optimal IOU to be $0.9 + 0.3 + 0.3 = 1.5$. But as the matrix get larger, the computational cost will increase drastically. The Hungarian Algorithm can solve the problem more efficiently using the four steps shown in fig. 4 and 5.

IOU tracking matrix, M

	Track1	Track2	Track3
Det 1	0.5	0.3	0.2
Det 2	0.3	0.2	0.3
Det 3	0.9	0.4	0.5

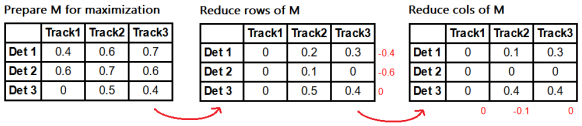
Brute force

	D1 T2	D2 T2	D2 T3
D1 T1	-	1.2	1.2
D2 T1	1.1	-	0.9
D3 T1	**1.5**	1.3	-

Figure 3: Illustation showing IOU tracking matrix M and the optimal IOU achieved through brute force.

1. First, we take the difference of each cell value in relation to the largest value in M, (0.9). This is because we are using IOU and therefore want to find the maximum optimal combination. This step is omitted if we want to find the minimum optimal combination.

2. Reduce the rows and columns by the minimum value in that row/column.



	Track1	Track2	Track3
Det 1	0.4	0.6	0.7
Det 2	0.6	0.7	0.6
Det 3	0	0.5	0.4

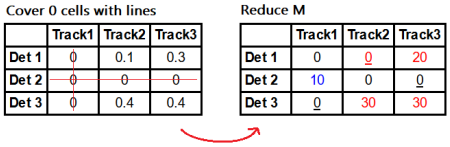
	Track1	Track2	Track3	
Det 1	0	0.2	0.3	-0.4
Det 2	0	0.1	0	-0.6
Det 3	0	0.5	0.4	0

	Track1	Track2	Track3
Det 1	0	0.1	0.3
Det 2	0	0	0
Det 3	0	0.4	0.4
	0	-0.1	0

Figure 4: Illustation showing maximization prepared M being reduced along rows and cols.

3. Cover the zero elements by the minimum number of horizontal and vertical lines. If the amount of lines is less than the size of the square array (3) move to next step.

4. Take the minimum uncovered element, subtract it from all the uncovered elements and add it to elements covered by two lines.



	Track1	Track2	Track3
Det 1	0	0.1	0.3
Det 2	0	0	0
Det 3	0	0.4	0.4

	Track1	Track2	Track3
Det 1	0	0	20
Det 2	10	0	0
Det 3	0	30	30

Figure 5: Illustration showing minimum amount of lines over reduced M and final result.

The result can now easily be observed to be $0.9 + 0.3 + 0.3 = 1.5$, by following the zero elements in the reduced matrix. This is how the Hungarian algorithm is used for data association and ID attribution - by effectively finding the optimal path.

2.5. The YOLO Detector

YOLO is a real-time detection model used to input detections to many tracking methods because of its SOTA performance. YOLOv8 is brand new and is the culmination of many advancements in the field of detection, since the original model [10]. A full illustration of its architecture is seen on fig. 6 [11]. This section relies heavily on the reader's knowledge of concepts from deep learning, as discussing all aspects of the YOLO architecture would be a paper in itself.

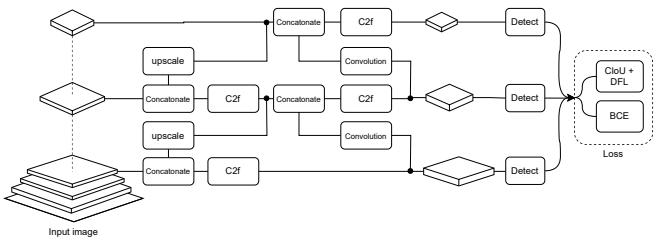


Figure 6: A conceptual illustration of the YOLOv8 model. Zoom in for details.

2.5.1. The Backbone

FPN: The YOLO family of models employs a Feature Pyramid Network as their backbone, which is a model that stores multiscale feature representations of an image. It combines features from different levels to enable scale-invariant object detection and recognition, while capturing contextual information. This approach improves localization accuracy. The same could be achieved by processing the raw images at different scales, but this is naturally more computationally costly. The feature pyramid is made up of *Conv*-blocks and `C2f`-blocks.

Conv-blocks: Are made up of a convolution layer to extract features, a batch normalization to handle training gradient issues, and an activation function to account for non-linearity.

SiLU: YOLO uses SiLU instead of ReLU as its activation function, which has better differentiability that helps prevent the "dying ReLU" problem and enables better gradient flow. SiLU has specifically been shown to improve generalization in deeper neural networks [12].

C2f: The C2F block refers to "Cross Stage Partial Network Fusion", and is the block that merges features across the different levels in the FPN by employing skip connections and bottle-necking to compress features representations, while still keeping the most important features intact.

2.5.2. The Head

The most significant changes in the v8 model are in the head module, which has been changed from the original coupling structure to a decoupling one and is now anchor-free.

The problem with anchors: One can imagine two cases, where relying on anchors become problematic.

- A single anchor may contain multiple identities (different objects of the same class), which would make its feature-embedding a mix of the two identities.
- Multiple anchors contain a single identity, which means that multiple feature-embeddings map to the same identity.

Traditionally, this is handled only by non-maximum suppression, which is not always effective. Going anchor-free means that the detector outputs centers and sizes of objects, meaning that it does not have this problem. The recent FairMOT tracker employs the same strategy [13].

3. The Tracking Algorithms

With the basic theory in place, we can now discuss our chosen state-of-the-art approaches to multi-object tracking.

3.1. StrongSORT

This approach builds on top of DeepSORT [14] which builds on SORT [15]. SORT is short for "Simple Online Real-time Tracking". It consists of four stages: detection, estimation, association and track identity creation. What sets apart tracking from simple detection is the estimation and association stages. Kalman filtering estimates the object's position and the Hungarian algorithm assigns the respective ID's (association).

As the name suggests, DeepSORT introduces deep learning

into the SORT algorithm by adding an appearance descriptor. This reduces inefficiencies caused by identity switches, which is when the tracker assigns a new ID to an already tracked object. An appearance descriptor is a feature vector that describes the appearance of an object by encoding its color, texture, and shape information. In an effort to reach SOTA, the fundamental parts of DeepSORT were upgraded using the latest technology. These parts include aspects like detection, feature embedding and trajectory association. The result is StrongSORT [2].

Figure 7 shows the overarching differences between StrongSORT and DeepSORT. StrongSORT uses a simple CNN whereas StrongSORT uses BoT+ReSNeSt50 as a backbone. It also uses the Enhanced Correlation Coefficient (ECC) to compensate for camera motion by aligning frames with image registration. This is done before it is fed to the more advanced NSA-Kalman Filter.

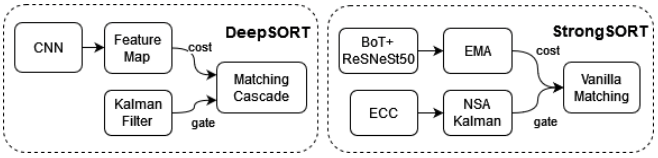


Figure 7: Figure illustrating the similarity between DeepSORT and StrongSORT [2]. The StrongSORT architecture is comprised of state-of-the-art technology.

3.2. DEEP OC-SORT

Deep OC-SORT is a brand-new tracking algorithm, at the time of writing, which builds upon OC-SORT [16], which in turn builds on SORT [15]. It is currently ranked first for the MOT-Challenge data-sets.

The problem with assuming linear motion: As we've discussed previously, all our tracking methods rely on the Kalman filter, which means that they must assume that objects move linearly, i.e. without much acceleration. This is clearly not always the case in real scenarios. Moreover, when there is no measurement available to update Kalman filter parameters, i.e. when occlusion occurs, the standard approach is to use the priori state estimations for posteriori update. This can lead to error accumulation. Imagine a video in which a car is accelerating to overtake a truck, but the truck temporarily hides the car from view. OC-SORT upon which Deep OC-SORT is based, aims to remedy this problem.

OC-SORT still uses a basic Kalman filter, but attempts to fix the noise accumulated during occlusion. Instead of relying only on the linear state estimate, it uses object observations to compute a virtual trajectory over the occlusion period. This allows more time steps to correct errors. OC-SORT proposes three remedies:

- **ORU**: Observation-centric re-update. ORU is triggered when a lost track is re-associated, and replaces the past estimations with better data which is retrieved from a virtual trajectory. The virtual trajectory starts at the track's last seen observation in it's history (z_{t_1}) and ends with the last association (z_{t_2}) which is the most recent observation. To generate the trajectory, it assumes a constant velocity and perform steps in the following fashion:

$$\hat{z}_t = z_{t_1} + \frac{t - t_1}{t_2 - t_1}(z_{t_2} - z_{t_1}) \tag{10}$$

  Where $t_1 < t < t_2$. The time step size as a hyper-parameter, which is tunable. OCR incorporates observations into the Kalman Filter in a manner that goes beyond tuning the filter parameters.

- **ORM**: Observation-centric momentum is the idea that the direction of motion of an object can be used to more accurately associate tracks. OCM incorporates observations into this estimation. The direction is given by:

$$\theta = \arctan\left(\frac{v_1 - v_2}{u_1 - u_2}\right) \tag{11}$$

  where (u_1, v_1) and (u_2, u_2) are two different bounding box positions. OCM keeps track of two angles. θ_{track} links two observations on a track, i.e. previous track observations are linked with every new detection. Each track forms a θ_{track} with every new detection. $\theta_{intention}$ links past a observation with a new observation, and each track only has one $\theta_{intention}$. These are compared to get a measure of direction, which the authors refer to it as the *consistency cost*. This cost is added to the traditional IoU cost for a more theoretically more accurate association.

$$\Delta\theta = |\theta_{track} - \theta_{intention}| \tag{12}$$

- **OCR**: Observation-centric recovery is a linear heuristic that may help recover a lost track by using it's last known observation. It associates the last known observation of each unmatched tracklet to all unmatched observations by an IoU score that occurs after the primary association steps.

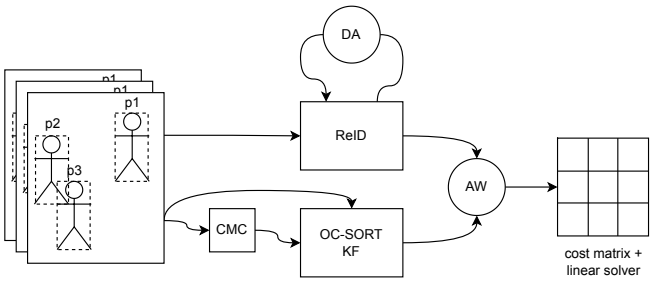


Figure 8: Simplified illustration of the Deep OC-SORT archtecture.

Essentially, ORU and OCM are remedies for tracking issues under occlusion or non-linear motion. OCR is a heuristic that attempts to prevent lost tracks with a secondary association.

Deep OC-SORT introduces **CMC**: Camera motion compensation, which learns how the camera is moving and transforms detections with a translation- and a rotation-matrix. OC-SORT essentially assumed that the camera is stationary, which is not always the case.

It also introduces **dynamic appearance**. Previously, an exponential moving average across frames was used to describe the feature embeddings of a tracklet. This requires a weighting factor to adjust the ratio of the embedding from previous and current time step. Dynamic appearance means that that ratio is updated on a per-frame-basis, depending on the detector confidence. This flexible factor allows selectively incorporating appearance information into a track's model only in good situations, i.e. sharp, non-occluded frames.

3.3. BoT-SORT

BoT-SORT is another tracking algorithm which introduces modifications to the already SOTA-performing ByteTrack algorithm. These all aim to solve the problems that have been discussed in the section on Deep OC-SORT, i.e. the linear motion assumption, camera-stationary assumption, etc. We will therefore focus on the most important component, which is the ByteTrack algorithm.

ByteTrack is the addition of an algorithm called BYTE on top of a detector, specifically YOLOX, which is an anchor-free forked version of YOLO [17]. We will be employing YOLOv8, i.e. the official version of YOLO, which also moved to anchor-free detections.

The problem with discarding low-scoring detections: Most object detection algorithms completely ignore bounding boxes with low confidence values to improve the rate of true positives (TP), but this is a patchwork solution. The objects with low scores, e.g. occluded objects, are simply thrown away, which brings non-negligible false negatives and fragmented trajectories. To solve this problem, BYTE performs tracking by associating *almost every* detection box, regardless of score, but separating them into high scores and low scores.

High-scoring detection boxes: Firstly, it associates the high score detection boxes to tracklets (tracked objects), which is standard practice. The first association is performed between the high score detection boxes and all the tracklets (including the unmatched). Similarity is computed by either by the IoU or the Re-ID feature distances between the high-scoring detection boxes and the predicted box of tracklets. The prediction is done using a Kalman filter. The Hungarian Algorithm finishes the matching based on the similarity.

Some tracklets get unmatched, because they do not match to an appropriate high score detection box, which usually happens when occlusion, motion blur or even apparent size-changes occurs.

Low-scoring detection boxes: This step is what sets ByteTrack apart from other methods. Falsely unmatched tracklets are recovered in this second association stage, and the background is simultaneously filtered out. The second association is performed between the low score detection boxes and the remaining tracklets. Unmatched low score detection boxes are considered background and discarded. The author found that IoU score alone was important for the second similarity calculation, because the low score detection boxes usually contains severe occlusion or motion blur and appearance features are not reliable. For "rebirthing" tracklets, it is necessary for the long-range association to preserve their identity. Only when it is not seen for more than a certain number of frames, i.e. 30, it is discarded.

BYTE is relatively simple and also highly flexible. For example, BYTE may be combined with FairMOT, simply by including Re-ID features is added into the first association. We argue that the principle of using almost all detection boxes can also be implemented in other tracking algorithms fairly easily. This is what the authors of BoT-SORT have done.

3.4. Alternative: FairMOT

Most tracking-by-detection MOTs excel at the detection task, but struggle with the re-identification task, which is due to three reasons:

1. Traditional trackers rely on anchors, which introduce ambiguities during the re-ID feature training or extraction. YOLOv8 does not use anchors, however, and is therefore no longer an issue.
2. Feature-sharing between the detection- and re-identification-task, which assumes that both tasks rely on the same features and treating re-ID'ing as a secondary task, meaning it depends on the primary detection task, which affects overall performance. It's known as the cascading effect.

The problem with tracking-by-detection: Fundamentally, the job of the detector is to distinguish between classes, for which it needs high-level features. The job of the re-identifier is to distinguish between identities of the same class, so it needs low-level features. For the traditional approach of "detection first, identification second", only high-level features from the detector are passed to the re-identifier. FairMOT uses a network called DLA-34 as seen in fig. 9, which is a combination of the backbone ResNet-34 [18] and an enhanced version of Deep Layer Aggregation [19], which fuses features from multiple layers.

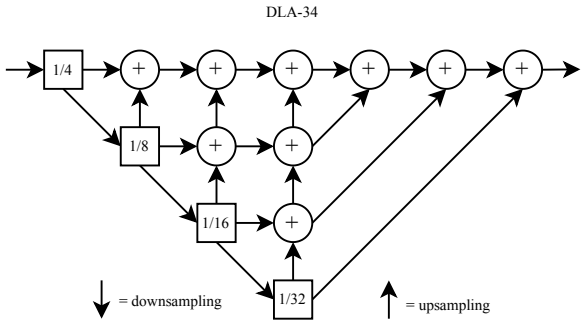


Figure 9: The DLA backbone of FairMOT. It's purpose is to fuse and therefore carry through lower-level features.

The FairMOT Architecture: Combining the solutions to the fundamental issues that were discussed, we arrive at the architecture for FairMOT, which may be viewed as the model-stage and association-stage, as seen with most single-shot-trackers.

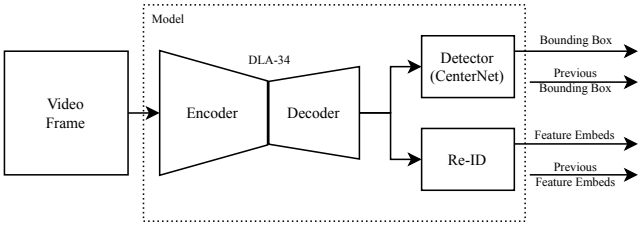


Figure 10: The model- or detection-stage of FairMOT, where most of the innovation occurs.

The association stage compares the current detections and

feature embeddings to their previous counterparts. A simple cosine distance D_r quantifies similarity between features. The cosine distance is chosen because the magnitude of the feature embedding vector is inconsequential. Mahalanobis distance D_m quantifies the distance between the current bounding boxes and its Kalman estimate. These are fused as:

$$D = \lambda D_r + (1 - \lambda) D_m \tag{13}$$

Finally, the Hungarian algorithm is used to match and retain as many identities as it can. For any unsuccessful matching, the estimated and current bounding boxes are compared using their IoU overlap. If within a certain margin, it is matched to an existing ID, otherwise a new ID is created.

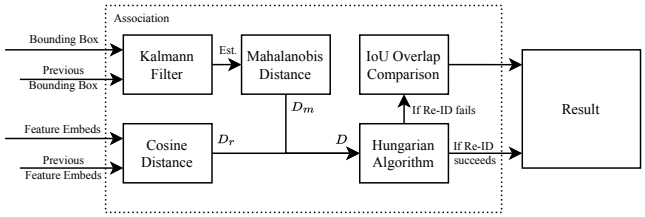


Figure 11: The association-stage of FairMOT. This is very similar to traditional association stages.

FairMOT represents a new tracking paradigm and looks promising and seemingly achieves good MOTA performance, although not SOTA yet. We intended to replicate the original author's results, but our attempts at running the code-base from their official repository were unsuccessful. We tried both Linux (with various kernels) and Windows, various versions and combinations of CUDA and python, and even inquiring into their exact testing environment. After dedicating weeks, we concluded that it was not possible, unless they provided a requirements.txt file along with a detailed description of their GPU setup, which they seemed unwilling to do. We also attempted to replicate the results of the StrongSORT paper directly, but also gave up on running their code for the same reasons. Additionally, we planned to experiment with TransMOT [20], which is tracking built on the popular transformer architecture [21], specifically the DETR model [22], but no code was available for testing from the TransMOT team.

4. Implementation

Accompanying this paper is a repository, which describes exactly how we achieved our performance measurements. It is designed to enable easy replication by the reader. As we were not able to run the code that was provided by the original authors of our selected tracking algorithms, we chose to tweak and integrate 3rd party implementations [23], and connecting them to a state-of-the-art YOLO-detector, namely YOLOv8x. The repository is set up such that creating and evaluating a tracking pipeline is simply a matter of specifying two components in a terminal:

1. **Detector**: Which of the YOLO models should act as the detector. There are several versions, differing in size and some having instance segmentation versions. These models were retrieved from the official YOLO repositories, pre-trained on the COCO dataset.
2. **Tracker**: Either StrongSORT, ByteTrack, BoT-SORT or Deep-OC-SORT.

To demo a tracking stack, simply follow the README in our repository, which will also provide detailed instructions on how to setup a virtual environment in which to run demos and evaluations.

The authors of the original papers for StrongSORT, BoT-SORT and Deep OC-SORT, as well as their respective predecessors, all have used different detectors to feed their tracking algorithms, meaning the detector most likely have had an influence on the overall performance scores, obscuring the true performance of the tracking algorithm. As we will be using the exact same detector for all of our measurements, regardless of dataset and tracking algorithm, our results should provide a more fair comparison of the tracking algorithms by themselves. This is only possible, because we have chosen to focus on the tracking-by-detection methodology.

For an ideal comparison of the tracking algorithms, the detector should be flawless, i.e. perform with 100% accuracy. As this is not possible, we will be using an approximate, which we have chosen to be the YOLOv8x model, i.e. the largest and most accurate model that Ultralytics (the official team behind YOLO) provides.

5. Results

We chose to evaluate on the MOT17 [24] dataset, as this seemingly is the most widely tested-on datasets in the literature. We utilize a very popular repository for evaluation that is specifically designed for the MOT-Challenge datasets [25], which was designed by the authors of the HOTA metric.

MOT17	Strong	BoT	Deep OC	Unit
HOTA	38.785	**41.968**	38.957	%
DetA	37.405	**40.983**	37.099	%
AssA	40.512	**43.352**	41.191	%
LocA	83.730	83.649	**84.123**	%
MOTA	39.409	**41.530**	38.574	%
MOTP	82.054	82.075	**82.585**	%
IDF1	43.236	**48.763**	45.021	%
IDSw	**645**	2112	1804	Total
IDTP	37016	**43616**	38146	Total
IDFP	21914	22977	**19016**	Total
Speed	∼**250**	∼400	∼300	ms/img

Table 1: A table summarizing the performance of each tracking algorithm. For exhaustive summaries of performance, please visit the "saved" folder in our repository. **note**: Speed i.e. processing time per frame, is including detection-time by YOLOv8x. Processing time becomes real-time, i.e. >30fps with the smaller YOLO models. Evaluation was done on Nvidia 1050 Ti.

The full MOT17 dataset consists of several types of videos, and as is evident when viewing the clips, these present different challenges.

Our measurements, e.g. HOTA, DetA, AssA and LocA, are all averaged across 7 individual video-clips, so to understand why one tracker may fail where another excels, we looked at the performance on clip-by-clip basis.

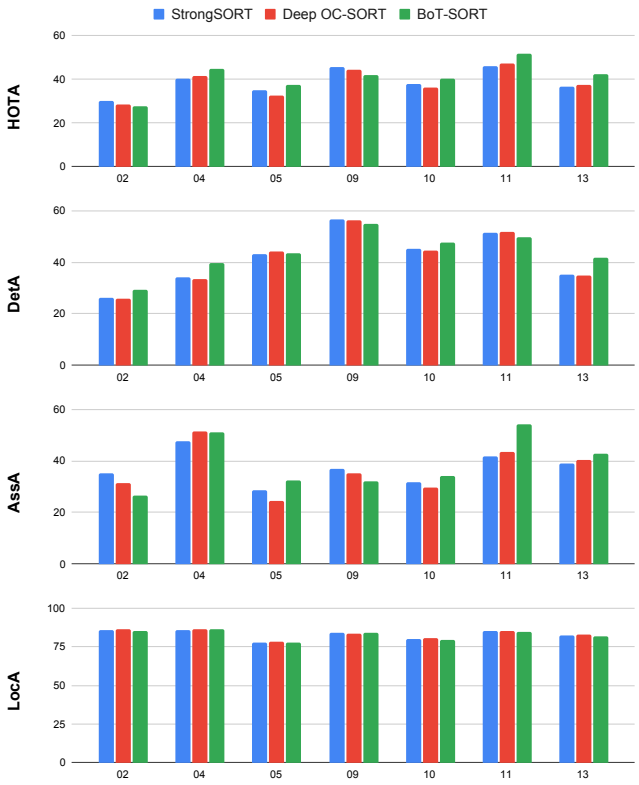


Figure 12: Plots showing performance measurements for each individual video clip in the MOT17 dataset.

- MOT17-02: People walking around a large square. Mid viewpoint, long-range visibility and stationary camera.
- MOT17-04: Pedestrian street at night, elevated viewpoint and stationary camera.
- MOT17-05: Street scene from a moving platform. Low viewpoint and short-range visibility.
- MOT17-09: A pedestrian street scene filmed from a low angle containing reflections. Stationary camera.
- MOT17-10: A pedestrian scene filmed at night by a moving camera.
- MOT17-11: Forward moving camera in a busy shopping mall containing reflections.
- MOT17-13: Filmed from a bus on a busy intersection.

Complete performance measurements for the MOT16 and MOT20 datasets are also available in our repository, but since they show the exact same tendencies, we chose to omit them from the report.

6. Conclusion

As mentioned in the results section, to get the best direct comparison between the tracking algorithms, their shared detector should perform near-flawlessly, as one tracking approach may rely more heavily on accurate detections than others. Had we had more time and resources, we would have trained the YOLO detector on the MOT-Challenge directly, before testing the entire tracking stacks. The YOLO detector was instead trained on the COCO dataset [26]. This may explain the consistent drop in scores across all metrics when compared to the original author's findings. However, this does not affect the validity of our direct tracker comparison, as the conditions are the same for all three algorithms.

Overall, BoT-SORT seems to be somewhat superior in accuracy, albeit marginally slower than the two others. This may be down to the fact that it builds on ByteTrack and therefore has more low-scoring detection boxes to process in each frame. Deep OC-SORT has noticeably fewer false positives than the other two, where BoT-SORT seems to be a little more aggressive at detecting. This contributes to the higher amount of ID-switching. StrongSORT seems to be the fastest of the three algorithms, but also the least accurate. Looking at the differences in performance for each scene in MOT17 separately, it makes sense, that the LocA score is near-identical for all the trackers, as this largely comes down to the performance of the detector stage, which is the same.

Interestingly, BoT-SORT seems to be under-performing in the associating task only on MOT17-09, which upon further inspection is the only scene that contain reflections that are labeled as valid detections, and MOT17-02, in which detections are strongly grouped together and overlapping. MOT17-11, in which BoT-SORT seems to perform much better on the association, also contain many reflections, but these are not labeled as valid detections, as the actual objects are also fully present in the view. The datasets captured with a camera mounted on a moving platform are MOT17-05, -10, -11 and -13. BoT-SORT performs noticeably better on the association task in these portions of the dataset, which translates to better HOTA score. This makes sense, as camera motion compensation is built in to the algorithm.

In summary, our findings suggest that BoT-SORT with ByteTrack is direction in which to go for best performance, which may explain why the official YOLO-team very recently started natively supporting BoT-SORT and vanilla ByteTrack with the YOLO detector and shipping those two algorithms with their official PIP package [27]. Furthermore, our HOTA rankings align with what is regarded as the official scoreboard, published by paperswithcode.com [28] at of the time of writing.

Contents

REFERENCES

[1] Fengwei Yu, Wenbo Li, Quanquan Li, Yu Liu, Xiaohua Shi, and Junjie Yan. POI: multiple object tracking with high performance detection and appearance feature. *CoRR*, abs/1610.06136, 2016.

[2] Yunhao Du, Zhicheng Zhao, Yang Song, Yanyun Zhao, Fei Su, Tao Gong, and Hongying Meng. Strongsort: Make deepsort great again. 2022.

[3] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. Bot-sort: Robust associations multi-pedestrian tracking. *arXiv preprint arXiv:2206.14651*, 2022.

[4] Gerard Maggiolino, Adnan Ahmad, Jinkun Cao, and Kris Kitani. Deep oc-sort: Multi-pedestrian tracking by adaptive re-identification. 2023.

[5] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The CLEAR MOT metrics. *EURASIP J. Image Video Process.*, 2008:1–10, 2008.

[6] Ergys Ristani, Francesco Solera, Roger S. Zou, Rita Cucchiara, and Carlo Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. *CoRR*, abs/1609.01775, 2016.

[7] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip H. S. Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. HOTA: A higher order metric for evaluating multi-object tracking. *CoRR*, abs/2009.07736, 2020.

[8] Prasanta Chandra Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49—55, 1936.

[9] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[10] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[11] RangeKing. Model structure of yolov8 detection models. jan 2023.

[12] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *CoRR*, abs/1702.03118, 2017.

[13] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. FairMOT: On the fairness of detection and re-identification in multiple object tracking. *International Journal of Computer Vision*, 129(11):3069–3087, sep 2021.

[14] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017.

[15] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016.

[16] Jinkun Cao, Jiangmiao Pang, Xinshuo Weng, Rawal Khirodkar, and Kris Kitani. Observation-centric sort: Rethinking sort for robust multi-object tracking. 2023.

[17] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box. 2022.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2015.

[19] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. 2019.

[20] Peng Chu, Jiang Wang, Quanzeng You, Haibin Ling, and Zicheng Liu. Transmot: Spatial-temporal graph transformer for multiple object tracking. *CoRR*, abs/2104.00194, 2021.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[22] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. 2020.

[23] Mikel Broström. Real-time multi-object, segmentation and pose tracking using Yolov8 with DeepOCSORT and LightMBN.

[24] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]*, March 2016. arXiv: 1603.00831.

[25] Jonathon Luiten. Trackeval - hota (and other) evaluation metrics for multi-object tracking (mot). may 2023.

[26] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[27] Ultralytics. Track - Tracking algorithms — docs.ultralytics.com. https://docs.ultralytics.com/modes/track/ Apr 2023. [Accessed 23-May-2023].

[28] Papers with Code. Papers with Code - MOT17 Benchmark (Multi-Object Tracking) — paperswithcode.com. https://paperswithcode.com/sota/multi-object-tracking-on-mot17?p=bot-sort-robust-associations-multi-pedestrian [Accessed 23-May-2023].