# Improving Kalman Filter-based Multi-Object Tracking in Occlusion and Non-linear Motion

Jinkun Cao

CMU-RI-TR-24-08

May 2024

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

# Abstract

Kalman filter (KF) based methods for multi-object tracking (MOT) assume that objects move linearly. While this assumption is acceptable for very short periods of occlusion, linear estimates of motion for prolonged time can be highly inaccurate. Moreover, when there is no measurement available to update Kalman filter parameters, the standard convention is to trust the priori state estimations for posteriori update. This leads to the accumulation of errors during a period of occlusion. The error causes significant motion direction variance in practice. In this work, we show that a basic Kalman filter can still obtain state-of-the-art tracking performance if proper care is taken to fix the noise accumulated during occlusion. Instead of relying only on the linear state estimate (i.e., estimation-centric approach), we use object observations (i.e., the measurements by object detector) to compute a virtual trajectory over the occlusion period to fix the error accumulation of filter parameters. This allows more time steps to correct errors accumulated during occlusion. We name our method **O**bservation-**C**entric **SORT** (OC-SORT). It remains Simple, Online, and Real-Time but improves robustness during occlusion and non-linear motion. Given off-the-shelf detections as input, OC-SORT runs at 700+ FPS on a single CPU. It achieves state-of-the-art on multiple datasets, including MOT17, MOT20, KITTI, head tracking, and especially DanceTrack where the object motion is highly non-linear. The code and models are available at https://github.com/noahcao/OC_SORT.

# Acknowledgments

I would like to thank my advisor, Professor Kris Kitani. He has been advising me patiently since the start of my Ph.D. during COVID. He is an inspiring, kind, and encouraging advisor, supportive to my ideas and plans. I would also like to express my thanks to other members of my committee, Professor Deva Ramanan, Professor David Held, and Zhengyi Luo. Their advice to this thesis, my presentation, and my Ph.D. study is valuable. Professor Deva Ramanan and Professor David Held have contributed many inspiring and helpful suggestions to improve this work since I first presented it to them for my Ph.D. speaking qualifier. My discussion with Professor David Held and his student Siddharth Ancha helped me to revise this project from a fundamental perspective of Kalman filters. Professor Deva Ramanan's suggestion about reviewing his previous work about detecting invisible people is also inspiring to me. All faculty members have provided generous advice about my presentation skills and thinking about the value of my project from a more fundamental, instead of a leaderboard-driven view. I thank Zhengyi Luo to be one of my committee members. Zhengyi is my closest peer to discuss academic questions and share ideas. We have been working together on multiple projects and I have learned a lot from Zhengyi, from physics-based vision to general research skills, for which I can not express my thankfulness enough. I also would like to thank Xinshuo Weng and Rawal Khirodkar for their advice to improve my presentation skills and detailed discussion when developing this project. I learned a lot from Xinshuo about how to do research in general and especially about detection and tracking. Rawal has been a close peer to me since the start of my Ph.D. from whom I also have learned a lot. Our consistent collaboration has helped me with multiple projects. By writing this master thesis, I am making a conclusion about my early Ph.D. research in multi-object tracking, which forms an important part of my overall Ph.D. goal and design. All the help and advice I received from these talented researchers contributed a lot to this stage of my Ph.D. journey. Finally, I want to thank all my friends and family who always support and love me.

# Contents

*When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.*

# List of Figures

# List of Tables

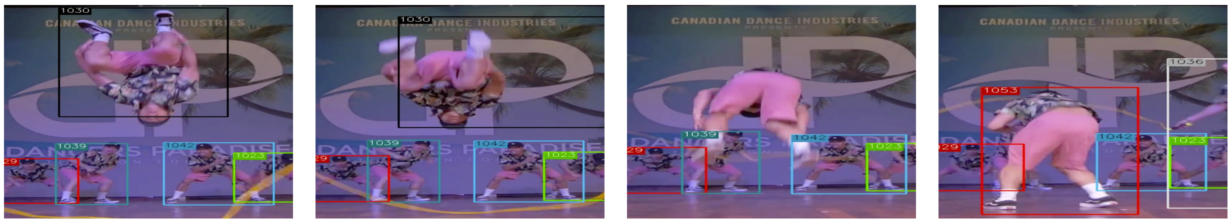
# Chapter 1

# Introduction

We aim to develop a motion model-based multi-object tracking (MOT) method that is robust to occlusion and non-linear motion. Most existing motion model-based algorithms assume that the tracking targets have a constant velocity within a time interval, which is called the linear motion assumption. This assumption breaks in many practical scenarios, but it still works because when the time interval is small enough, the object's motion can be reasonably approximated as linear. In this work, we are motivated by the fact that most of the errors from motion model-based tracking methods occur when occlusion and non-linear motion happen together. To mitigate the adverse effects caused, we first rethink current motion models and recognize some limitations. Then, we propose addressing them for more robust tracking performance, especially in occlusion.

As the main branch of motion model-based tracking, filtering-based methods assume a transition function to predict the state of objects on future time steps, which are called state "estimations". Besides estimations, they leverage an observation model, such as an object detector, to derive the state measurements of target objects, also called "observations". Observations usually serve as auxiliary information to help update the posteriori parameters of the filter. The trajectories are still extended by the state estimations. Among this line of work, the most widely used one is SORT [5], which uses a Kalman filter (KF) to estimate object states and a linear motion function as the transition function between time steps. However, SORT shows insufficient tracking robustness when the object motion is non-linear, and no observations are available when updating the filter posteriori parameters.

In this work, we recognize **three limitations** of SORT. First, although the high frame rate is the key to approximating the object motion as linear, it also amplifies the model's sensitivity to the noise of state estimations. Specifically, between consecutive frames of a high frame-rate video, we demonstrate that the noise of displacement of the object can be of the


(a) SORT

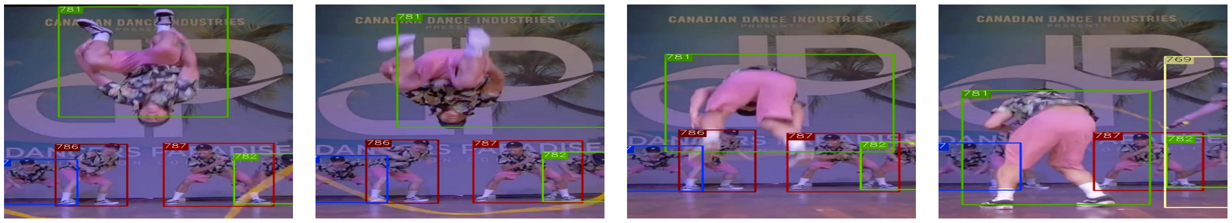

(b) The proposed OC-SORT

Figure 1.1: Samples from the results on DanceTrack [63]. SORT and OC-SORT use the same detection results. On the third frame, SORT encounters an ID switch for the backflip target while OC-SORT tracks it consistently.

same magnitude as the actual object displacement, leading to the estimated object velocity by KF suffering from a significant variance. Also, the noise in the velocity estimate will accumulate into the position estimate by the transition process. Second, the noise of state estimations by KF is accumulated along the time when there is no observation available in the update stage of KF. We show that the error accumulates very fast with respect to the time of the target object's being untracked. The noise's influence on the velocity direction often makes the track lost again even after re-association. Last, given the development of modern detectors, the object state by detections usually has lower variance than the state estimations propagated along time steps by a fixed transition function in filters. However, SORT is designed to prolong the object trajectories by state estimations instead of observations.

To relieve the negative effect of these limitations, we propose two main innovations in this work. First, we design a module to use object state observations to reduce the accumulated error during the track's being lost in a backcheck fashion. To be precise, besides the traditional stages of *predict* and *update*, we add a stage of *re-update* to correct the accumulated error. The *re-update* is triggered when a track is re-activated by associating to an observation after a period of being untracked. The *re-update* uses virtual observations on the historical time steps to prevent error accumulation. The virtual observations come from a trajectory generated using the last-seen observation before untracked and the latest observation re-activating this track as anchors. We name it *Observation-centric Re-Update (ORU)*.

Besides ORU, the assumption of linear motion provides the consistency of the object motion direction. But this cue is hard to be used in SORT's association because of the heavy noise in direction estimation. But we propose an observation-centric manner to incorporate the direction consistency of tracks in the cost matrix for the association. We name it *Observation-Centric Momentum (OCM)*. We also provide analytical justification for the noise of velocity direction estimation in practice.

The proposed method, named as **O**bservation-**C**entric **SORT** or **OC-SORT** in short, remains simple, online, real-time and significantly improves robustness over occlusion and non-linear motion. Our contributions are summarized as the following:

1. We recognize, analytically and empirically, three limitations of SORT, i.e.sensitivity to the noise of state estimations, error accumulation over time, and being estimation-centric;
2. We propose OC-SORT for tracking under occlusion and non-linear motion by fixing SORT's limitations. It achieves state-of-the-art performance on multiple datasets in an online and real-time fashion.

# Chapter 2

# Background

## 2.1 Motion Models

Many modern MOT algorithms [5, 15, 71, 79, 83] use motion models. Typically, these motion models use Bayesian estimation [41] to predict the next state by maximizing a posterior estimation. As one of the most classic motion models, Kalman filter (KF) [37] is a recursive Bayes filter that follows a typical predict-update cycle. The true state is assumed to be an unobserved Markov process, and the measurements are observations from a hidden Markov model [52]. Given that the linear motion assumption limits KF, follow-up works like Extended KF [60] and Unscented KF [35] were proposed to handle non-linear motion with first-order and third-order Taylor approximation. However, they still rely on approximating the Gaussian prior assumed by KF and require motion pattern assumption. On the other hand, particle filters [28] solve the non-linear motion by sampling-based posterior estimation but require exponential order of computation. Therefore, these variants of Kalman filter and particle filters are rarely adopted in the visual multi-object tracking and the mostly adopted motion model is still based on Kalman filter [5].

## 2.2 Multi-object Tracking

As a classic computer vision task, visual multi-object tracking is traditionally approached from probabilistic perspectives, e.g.joint probabilistic association [2]. And modern video object tracking is usually built upon modern object detectors [54, 56, 82]. SORT [5] adopts the Kalman filter for motion-based multi-object tracking given observations from deep detectors. DeepSORT [71] further introduces deep visual features [29, 59] into object association under the framework of SORT. Re-identification-based object association[50, 71, 80] has also become popular since then but falls short when scenes are crowded and objects are represented coarsely (e.g.enclosed by bounding boxes), or object appearance is not distinguishable. More recently, transformers [66] have been introduced to MOT [12, 47, 62, 77] to learn deep representations from both visual information and object trajectories. However, their performance still has a significant gap between state-of-the-art tracking-by-detection methods in terms of both accuracy and time efficiency.

# Chapter 3

# Method

## 3.1 Rethink the Limitations of SORT

In this section, we review Kalman filter and its widely used implementation for multi-object tracking, i.e.SORT [5]. We recognize some of their limitations, which are significant with **occlusion** and **non-linear object motion**. In this work, we are motivated to improve the accuracy and robustness of Kalman filter-based multi-object tracking by fixing these recognized limitations.

### 3.1.1 Preliminaries

**Kalman filter (KF)** [37] is a linear estimator for dynamical systems discretized in the time domain. KF only requires the state estimations on the previous time step and the current measurement to estimate the target state on the next time step. The filter maintains two variables, the posteriori state estimate $\mathbf{x}$, and the posteriori estimate covariance matrix $\mathbf{P}$. In the task of object tracking, we describe the KF process with the state transition model $\mathbf{F}$, the observation model $\mathbf{H}$, the process noise $\mathbf{Q}$, and the observation noise $\mathbf{R}$. At each step $t$, given observations $\mathbf{z}_t$, KF works in an alternation of *predict* and *update* stages:

$$\begin{aligned}
&\textit{predict}\begin{cases}\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}_t\hat{\mathbf{x}}_{t-1|t-1}\\ \mathbf{P}_{t|t-1} = \mathbf{F}_t\mathbf{P}_{t-1|t-1}\mathbf{F}_t^\top + \mathbf{Q}_t\end{cases},\\
&\textit{update}\begin{cases}\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^\top(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^\top + \mathbf{R}_t)^{-1}\\ \hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mathbf{H}_t\hat{\mathbf{x}}_{t|t-1})\\ \mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t|t-1}\end{cases}.
\end{aligned} \tag{3.1}$$

The stage of *predict* is to derive the state estimations on the next time step $t$. Given a measurement of target states on the next step $t$, the stage of *update* aims to update the posteriori parameters in KF. Because the measurement comes from the observation model $\mathbf{H}$, it is also called "observation" in many scenarios.

**SORT** [5] is a multi-object tracker built upon KF. The KF's state $\mathbf{x}$ in SORT is defined as $\mathbf{x} = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^\top$, where $(u,\ v)$ is the 2D coordinates of the object center in the image. $s$ is the bounding box scale (area) and $r$ is the bounding box aspect ratio. The aspect ratio $r$ is assumed to be constant. The other three variables, $\dot{u}$, $\dot{v}$ and $\dot{s}$ are the corresponding time derivatives. The observation is a bounding box $\mathbf{z} = [u, v, w, h, c]^\top$ with object center position $(u, v)$, object width $w$, and height $h$ and the detection confidence $c$ respectively. SORT assumes linear motion as the transition model $\mathbf{F}$ which leads to the state estimation as

$$u_{t+1} = u_t + \dot{u}_t \Delta t, \quad v_{t+1} = v_t + \dot{v}_t \Delta t. \tag{3.2}$$

To leverage KF (Equation (3.1)) in SORT for visual MOT, the stage of *predict* corresponds to estimating the object position on the next video frame. And the observations used for the update stage usually come from a detection model. The update stage is to update Kalman filter parameters and does not directly edit the tracking outcomes.

When the time difference between two steps is constant during the transition, e.g., the video frame rate is constant, we can set $\Delta t = 1$. When the video frame rate is high, SORT works well even when the object motion is non-linear globally, (e.g.dancing, fencing, wrestling) because the motion of the target object can be well approximated as linear within short time intervals. However, in practice, observations are often absent on some time steps, e.g.the target object is occluded in multi-object tracking. In such cases, we cannot update the KF parameters by the update operation as in Equation (3.1) anymore. SORT uses the priori estimations directly as posterior. We call this **"dummy update"**, namely

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1}, \mathbf{P}_{t|t} = \mathbf{P}_{t|t-1}. \tag{3.3}$$

The philosophy behind such a design is to trust estimations when no observations are available to supervise them. We thus call the tracking algorithms following this scheme "estimation-centric". However, we will see that this estimation-centric mechanism can cause trouble when non-linear motion and occlusion happen together.

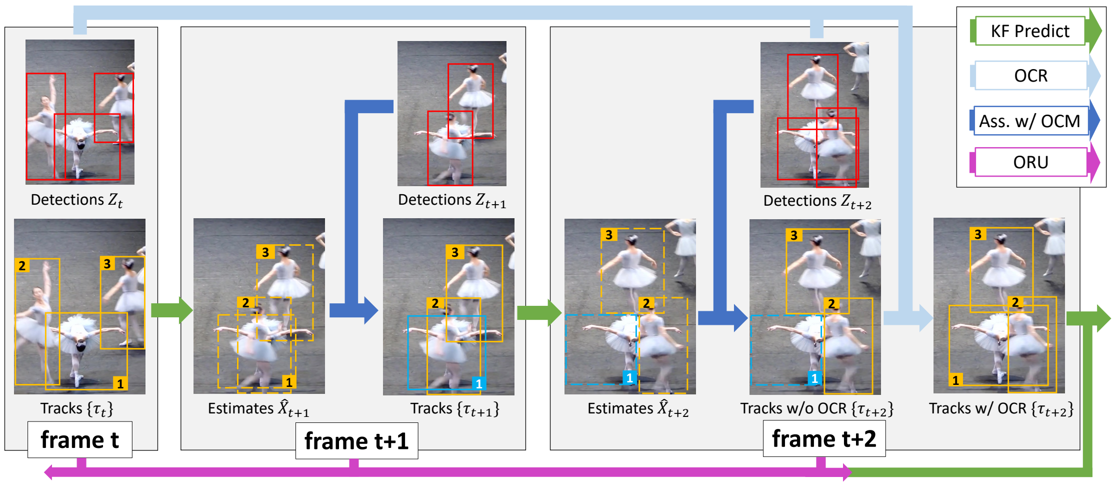


Figure 3.1: The pipeline of our proposed OC-SORT. The red boxes are detections, orange boxes are active tracks, blue boxes are untracked tracks, and dashed boxes are the estimates from KF. During association, OCM is used to add the velocity consistency cost. The target #1 is lost on the frame t+1 because of occlusion. But on the next frame, it is recovered by referring to its observation of the frame t by OCR. It being re-tracked triggers ORU from t to t+2 for the parameters of its KF.

## 3.1.2 Limitations of SORT

In this section, we identify three main limitations of SORT which are connected. This analysis lays the foundation of our proposed method.

### Sensitive to State Noise

Now we show that SORT is sensitive to the noise from KF's state estimations. To begin with, we assume that the estimated object center position follows $u \sim \mathcal{N}(\mu_u, \sigma_u^2)$ and $v \sim \mathcal{N}(\mu_v, \sigma_v^2)$, where $(\mu_u, \mu_v)$ is the underlying true position. Then, if we assume that the state noises are independent on different steps, by Equation (3.2), the object speed between two time steps, $t \longrightarrow t + \Delta t$, is

$$\dot{u} = \frac{u_{t+\Delta t} - u_t}{\Delta t}, \qquad \dot{v} = \frac{v_{t+\Delta t} - v_t}{\Delta t}, \tag{3.4}$$

making the noise of estimated speed $\delta_{\dot{u}} \sim \mathcal{N}(0, \frac{2\sigma_u^2}{(\Delta t)^2})$, $\delta_{\dot{v}} \sim \mathcal{N}(0, \frac{2\sigma_v^2}{(\Delta t)^2})$. Therefore, a small $\Delta t$ will amplify the noise. This suggests that SORT will suffer from the heavy noise of velocity estimation on high-frame-rate videos. The analysis above is simplified from the reality. In pratice, velocity won't be determined by the state on future time steps. For a more strict analysis, please refer to Section 5.3.

Moreover, for most multi-object tracking scenarios, the target object displacement is only a few pixels between consecutive frames. For instance, the average displacement is 1.93 pixels and 0.65 pixels along the image width and height for the MOT17 [49] training dataset.

In such a case, even if the estimated position has a shift of only a single pixel, it causes a significant variation in the estimated speed. In general, the variance of the speed estimation can be of the same magnitude as the speed itself or even greater. This will not make a massive impact as the shift is only of few pixels from the ground truth on the next time step and the observations, whose variance is independent of the time, will be able to fix the noise when updating the posteriori parameters. However, we find that such a high sensitivity to state noise introduces significant problems in practice after being amplified by the error accumulation across multiple time steps when no observation is available for KF *update*.

**Temporal Error Magnification**

For analysis above in Equation (3.4), we assume the noise of the object state is i.i.d on different time steps (this is a simplified version, a more detailed analysis is provided in Section 5.3). This is reasonable for object detections but not for the estimations from KF. This is because KF's estimations always rely on its estimations on previous time steps. The effect is usually minor because KF can use observation in *update* to prevent the posteriori state estimation and covariance, i.e.$\hat{\mathbf{x}}_{t|t}$ and $\mathbf{P}_{t|t}$, deviating from the true value too far away. However, when no observations are provided to KF, it cannot use observation to update its parameters. Then it has to follow Equation (3.3) to prolong the estimated trajectory to the next time step. Consider a track is occluded on the time steps between $t$ and $t+T$ and the noise of speed estimate follows $\delta_{\dot{u}_t} \sim \mathcal{N}(0, 2\sigma_u^2)$, $\delta_{\dot{v}_t} \sim \mathcal{N}(0, 2\sigma_v^2)$ for SORT. On the step $t+T$, state estimation would be

$$u_{t+T} = u_t + T\dot{u}_t, \qquad v_{t+T} = v_t + T\dot{v}_t, \tag{3.5}$$

whose noise follows $\delta_{u_{t+T}} \sim \mathcal{N}(0, 2T^2\sigma_u^2)$ and $\delta_{v_{t+T}} \sim \mathcal{N}(0, 2T^2\sigma_v^2)$. So without the observations, the estimation from the linear motion assumption of KF results in a fast error accumulation with respect to time. Given $\sigma_v$ and $\sigma_u$ is of the same magnitude as object displacement between consecutive frames, the noise of final object position $(u_{t+T}, v_{t+T})$ is of the same magnitude as the object size. For instance, the size of pedestrians close to the camera on MOT17 is around $50 \times 300$ pixels. So even assuming the variance of position estimation is only 1 pixel, 10-frame occlusion can accumulate a shift in final position estimation as large as the object size. Such error magnification leads to a major accumulation of errors when the scenes are crowded.

**Estimation-Centric**

The aforementioned limitations come from a fundamental property of SORT that it follows KF to be estimation-centric. It allows *update* without the existence of observations and purely trusts the estimations. A key difference between state estimations and observations is

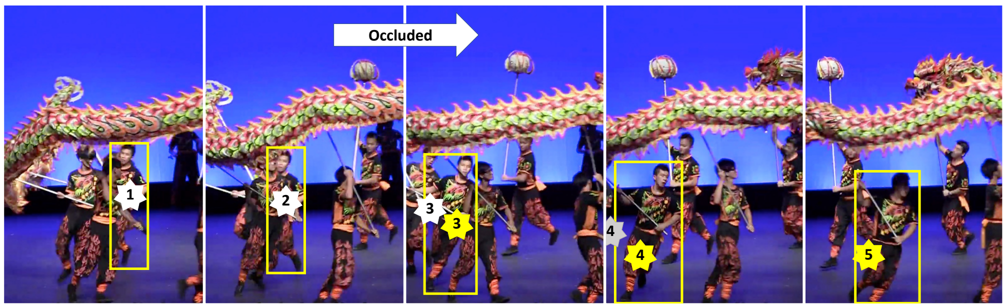


Figure 3.2: Example of how *Observation-centric Re-Update (ORU)* reduces the error accumulation when a track is broken. The target is occluded between the second and the third time step and the tracker finds it back at the third step. Yellow boxes are the state observations by the detector. White stars are the estimated centers without ORU. Yellow stars are the estimated centers fixed by ORU. The gray star on the fourth step is the estimated center without ORU and fails to match observations.

that we can assume that the observations by an object detector in each frame are affected by i.i.d. noise $\delta_{\mathbf{z}} \sim \mathcal{N}(0, \sigma'^2)$ while the noise in state estimations can be accumulated along the hidden Markov process. Moreover, modern object detectors use powerful object visual features [56, 59]. It makes that, even on a single frame, it is usually safe to assume $\sigma' < \sigma_u$ and $\sigma' < \sigma_v$ because the object localization is more accurate by detection than from the state estimations through linear motion assumption. Combined with the previously mentioned two limitations, being estimation-centric makes SORT suffer from heavy noise when there is occlusion and the object motion is not perfectly linear.

## 3.2 Observation-Centric SORT

In this section, we introduce the proposed *Observation-Centric SORT (OC-SORT)*. To address the limitations of SORT discussed above, we use the momentum of the object moving into the association stage and develop a pipeline with less noise and more robustness over occlusion and non-linear motion. The key is to design the tracker as **observation-centric** instead of **estimation-centric**. If a track is recovered from being untracked, we use an *Observation-centric Re-Update (ORU)* strategy to counter the accumulated error during the untracked period. OC-SORT also adds an *Observation-Centric Momentum (OCM)* term in the association cost. Please refer to Algorithm 1 for the pseudo-code of OC-SORT. The pipeline is shown in Figure 3.1. See the pseudo-code of OC-SORT in Algorithm 1.

### 3.2.1 Observation-centric Re-Update (ORU)

In practice, even if an object can be associated again by SORT after a period of being untracked, it is probably lost again because its KF parameters have already deviated far away from the correct due to the temporal error magnification. To alleviate this problem, we propose *Observation-centric Re-Update (ORU)* to reduce the accumulated error. Once a track is associated with an observation again after a period of being untracked ("re-activation"), we backcheck the period of its being lost and re-update the parameters of KF. The re-update is based on "observations" from a virtual trajectory. The virtual trajectory is generated referring to the observations on the steps starting and ending the untracked period. For example, by denoting the last-seen observation before being untracked as $\mathbf{z}_{t_1}$ and the observation triggering the re-association as $\mathbf{z}_{t_2}$, the virtual trajectory is denoted as

$$\tilde{\mathbf{z}}_t = Traj_{\text{virtual}}(\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, t), t_1 < t < t_2. \tag{3.6}$$

Then, along the trajectory of $\tilde{\mathbf{z}}_t (t_1 < t < t_2)$, we run the loop of *predict* and *re-update*. The *re-update* operation is

$$\textit{re-update} \begin{cases} \mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^\top(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^\top + \mathbf{R}_t)^{-1} \\ \hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\tilde{\mathbf{z}}_t - \mathbf{H}_t\hat{\mathbf{x}}_{t|t-1}) \\ \mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t|t-1} \end{cases} \tag{3.7}$$

As the observations on the virtual trajectory match the motion pattern anchored by the last-seen and the latest associated real observations, the update will not suffer from the error accumulated through the dummy update anymore. We call the proposed process *Observation-centric Re-Update*. It serves as an independent stage outside the *predict-update* loop and is triggered only a track is re-activated from a period of having no observations.

### 3.2.2 Observation-Centric Momentum (OCM)

In a reasonably short time interval, we can approximate the motion as linear. And the linear motion assumption also asks for consistent motion direction. But the noise prevents us from leveraging the consistency of direction. To be precise, to determine the motion direction, we need the object state on two steps with a time difference $\Delta t$. If $\Delta t$ is small, the velocity noise would be significant because of the estimation's sensitivity to state noise. If $\Delta t$ is big, the noise of direction estimation can also be significant because of the temporal error magnification and the failure of linear motion assumption. As state observations have no

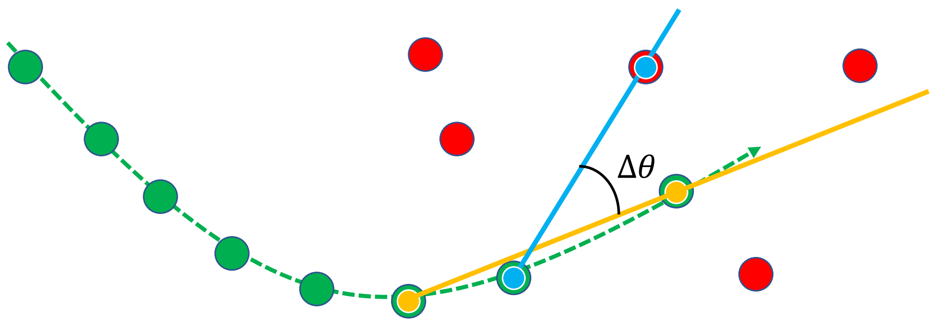


Figure 3.3: Calculation of motion direction difference in OCM. The green line indicates an existing track and the dots are the observations on it. The red dots are the new observations to be associated. The blue link and the yellow link form the directions of $\theta^{\text{track}}$ and $\theta^{\text{intention}}$ respectively. The included angle is the difference of direction $\Delta\theta$.

problem of temporal error magnification that state estimations suffer from, we propose to use observations instead of estimations to reduce the noise of motion direction calculation and introduce the term of velocity consistency to help the association.

With the new term, given $N$ existing tracks and $M$ detections on the new-coming time step, the association cost matrix is formulated as

$$C(\hat{\mathbf{X}}, \mathbf{Z}) = C_{\text{IoU}}(\hat{\mathbf{X}}, \mathbf{Z}) + \lambda C_v(\mathcal{Z}, \mathbf{Z}), \tag{3.8}$$

where $\hat{\mathbf{X}} \in \mathbb{R}^{N\times 7}$ is the set of object state estimations and $\mathbf{Z} \in \mathbb{R}^{M\times 5}$ is the set of observations on the new time step. $\lambda$ is a weighting factor. $\mathcal{Z}$ contains the trajectory of observations of all existing tracks. $C_{\text{IoU}}(\cdot,\cdot)$ calculates the negative pairwise IoU (Intersection over Union) and $C_v(\cdot,\cdot)$ calculates the consistency between the directions of i) linking two observations on an existing track ($\theta^{\text{track}}$) and ii) linking a track's historical observation and a new observation ($\theta^{\text{intention}}$). $C_v$ contains all pairs of $\Delta\theta = |\theta^{\text{track}} - \theta^{\text{intention}}|$. In our implementation, we calculate the motion direction in radians, namely $\theta = \arctan(\frac{v_1 - v_2}{u_1 - u_2})$ where $(u_1, v_1)$ and $(u_2, v_2)$ are the observations on two different time steps. The calculation is also illustrated in Figure 3.3.

Following the assumptions of noise distribution mentioned before, we can derive a closed-form probability density function of the distribution of the noise in the direction estimation. The derivation is explained in detail in Section 5.1. By analyzing the property of this distribution, we reach a conclusion that, under the linear-motion model, the scale of the noise of direction estimation is negatively correlated to the time difference between the two observation points, i.e.$\Delta t$. This suggests increasing $\Delta t$ to achieve a low-noisy estimation of $\theta$. However, the assumption of linear motion typically holds only when $\Delta t$ is small enough. Therefore, the choice of $\Delta t$ requires a trade-off.

Besides ORU and OCM, we also find it empirically helpful to check a track's last presence to recover it from being lost. We thus apply a heuristic *Observation-Centric Recovery (OCR)* technique. OCR will start a second attempt of associating between the last observation of unmatched tracks to the unmatched observations after the usual association stage. It can handle the case of an object stopping or being occluded for a short time interval.

By combining the proposed components upon the standard SORT [5] algorithm, we finally implemented the Observation-Centric SORT, i.e.OC-SORT. The overall process of the proposed algorithm in shown in Algorithm 1.

**Algorithm 1** Pseudo-code of OCSORT.

```
Input: Detections $\mathcal{Z} = \{\mathbf{z}_k^i | 1 \le k \le T, 1 \le i \le N_k\}$; Kalman Filter KF; threshold to remove untracked tracks $t_{\text{expire}}$
Output: The set of tracks $\mathcal{T} = \{\tau_i\}$
Initialization: $\mathcal{T} \leftarrow \emptyset$ and KF;
for timestep $t \leftarrow 1 : T$ do
    /* Step 1:  match track prediction with observations */
    $\mathbf{Z}_t \leftarrow [\mathbf{z}_t^1, ..., \mathbf{z}_t^{N_t}]^\top$ /* Obervations */
    $\hat{\mathbf{X}}_t \leftarrow [\hat{\mathbf{x}}_t^1, ..., \hat{\mathbf{x}}_t^{|\mathcal{T}|}]^\top$ from $\mathcal{T}$ /* Estimations by KF.predict */
    $\mathcal{Z} \leftarrow$ Historical observations on the existing tracks
    $C_t \leftarrow C_{\text{IoU}}(\hat{\mathbf{X}}_t, \mathbf{Z}_t) + \lambda C_v(\mathcal{Z}, \mathbf{Z}_t)$ /* Cost Matrix with OCM term */
    Linear assignment by Hungarians with cost $C_t$
    $\mathcal{T}_t^{\text{matched}} \leftarrow$ tracks matched to an observation
    $\mathcal{T}_t^{\text{remain}} \leftarrow$ tracks not matched to any observation
    $\mathbf{Z}_t^{\text{remain}} \leftarrow$ observations not matched to any track

    /* Step 2:  perform OCR to find lost tracks back */
    $\mathbf{Z}^{\mathcal{T}_t^{\text{remain}}} \leftarrow$ last matched observations of tracks in $\mathcal{T}_t^{\text{remain}}$
    $C_t^{\text{remain}} \leftarrow C_{\text{IoU}}(\mathbf{Z}^{\mathcal{T}_t^{\text{remain}}}, \mathbf{Z}_t^{\text{remain}})$
    Linear assignment by Hungarians with cost $C_t^{\text{remain}}$
    $\mathcal{T}_t^{\text{recovery}} \leftarrow$ tracks from $\mathcal{T}_t^{\text{remain}}$ and matched to observations in $\mathbf{Z}^{\mathcal{T}_t^{\text{remain}}}$
    $\mathbf{Z}_t^{\text{unmatched}} \leftarrow$ observations from $\mathbf{Z}^{\mathcal{T}_t^{\text{remain}}}$ that are still unmatched to tracks
    $\mathcal{T}_t^{\text{unmatched}} \leftarrow$ tracks from $\mathcal{T}_t^{\text{remain}}$ that are still unmatched to observations
    $\mathcal{T}_t^{\text{matched}} \leftarrow \{\mathcal{T}_t^{\text{matched}}, \mathcal{T}_t^{\text{recovery}}\}$

    /* Step 3:  update status of matched tracks */
    for $\tau$ in $\mathcal{T}_t^{matched}$ do
        if $\tau.tracked = False$ then
            /* Perform ORU for track from untracked to tracked */
            $\mathbf{z}_{t'}^\tau, t' \leftarrow$ The last observation matched to $\tau$ and the time step
            Rollback KF parameters to $t'$
            /* Generate virtual observation trajectory */
            $\hat{\mathbf{Z}}_t^\tau \leftarrow [\hat{\mathbf{z}}_{t'+1}^\tau, ..., \hat{\mathbf{z}}_{t-1}^\tau]$
            Online smooth KF parameters along $\hat{\mathbf{Z}}_t^\tau$
        end
        $\tau.tracked = True$
        $\tau.untracked = 0$
        Append the new matched associated observation $\mathbf{z}_t^\tau$ to $\tau$'s observation history
        Update KF parameters for $\tau$ by $\mathbf{z}_t^\tau$
    end

    /* Step 4:  initialize new tracks and remove expired tracks */
    $\mathcal{T}_t^{new} \leftarrow$ new tracks generated from $\mathbf{Z}_t^{\text{unmatched}}$
    for $\tau$ in $\mathcal{T}_t^{unmatched}$ do
        $\tau.tracked = False$
        $\tau.untracked = \tau.untracked + 1$
    end
    $\mathcal{T}_t^{\text{reserved}} \leftarrow \{\tau | \ \tau \in \mathcal{T}_t^{\text{unmatched}}$ and $\tau.untacked < t_{\text{expire}}\}$ /* remove expired unmatched tracks */
    $\mathcal{T} \leftarrow \{\mathcal{T}_t^{\text{new}}, \mathcal{T}_t^{\text{matched}}, \mathcal{T}_t^{\text{reserved}}\}$ /* Conclude */
end
$\mathcal{T} \leftarrow$ Postprocess($\mathcal{T}$) /* [Optional] offline post-processing */
Return: $\mathcal{T}$
```

# Chapter 4

# Experiments

## 4.1 Experimental Setup

**Datasets.** We evaluate our method on multiple multi-object tracking datasets including MOT17 [49], MOT20 [19], KITTI [26], DanceTrack [63] and CroHD [64]. MOT17 [49] and MOT20 [19] are for pedestrian tracking, where targets mostly move linearly, while scenes in MOT20 are more crowded. KITTI [26] is for pedestrian and car tracking with a relatively low frame rate of 10FPS. CroHD [64] is a dataset for head tracking in the crowd. DanceTrack [63] is a recently proposed dataset for human tracking. For the data in DanceTrack, object localization is easy, but the object motion is highly non-linear. Furthermore, the objects have a close appearance, severe occlusion, and frequent crossovers. Considering our goal is to improve tracking robustness under occlusion and non-linear object motion, we would emphasize the comparison on DanceTrack.

**Implementations.** For a fair comparison, we directly apply the object detections from existing baselines. For MOT17, MOT20, and DanceTrack, we use the publicly available YOLOX [25] detector weights by ByteTrack [79]. For KITTI [26], we use the detections from PermaTrack [65] publicly available in the official release[1]. For ORU, we generate the virtual trajectory during occlusion with the constant-velocity assumption. Therefore, Equation (3.6) is adopted as $\tilde{\mathbf{z}}_t = \mathbf{z}_{t_1} + \frac{t-t_1}{t_2-t_1}(\mathbf{z}_{t_2} - \mathbf{z}_{t_1}), t_1 < t < t_2$. For OCM, the velocity direction is calculated using the observations three time steps apart, i.e.$\Delta t = 3$. The direction difference is measured by the absolute difference of angles in radians. We set $\lambda = 0.2$ in Equation (3.8). Following the common practice of SORT, we set the detection confidence threshold at 0.4 for MOT20 and 0.6 for other datasets. The IoU threshold during association is 0.3.

**Metrics.** We adopt HOTA [44] as the main metric as it maintains a proper balance between

[1]https://github.com/TRI-ML/permatrack/

the accuracy of object detection and association [44]. We also emphasize AssA to evaluate the association performance. IDF1 is also used for association performance evaluation. Other metrics we report, such as MOTA, are highly related to detection performance.

Table 4.1: Results on MOT17-test with the private detections. ByteTrack and OC-SORT share detections.

| Tracker | HOTA↑ | MOTA↑ | IDF1↑ | FP($10^4$)↓ | FN($10^4$)↓ | IDs↓ | Frag↓ | AssA↑ | AssR↑ |
|---|---|---|---|---|---|---|---|---|---|
| FairMOT [80] | 59.3 | 73.7 | 72.3 | 2.75 | 11.7 | 3,303 | 8,073 | 58.0 | 63.6 |
| TransCt [75] | 54.5 | 73.2 | 62.2 | 2.31 | 12.4 | 4,614 | 9,519 | 49.7 | 54.2 |
| TransTrk [62] | 54.1 | 75.2 | 63.5 | 5.02 | 8.64 | 3,603 | 4,872 | 47.9 | 57.1 |
| GRTU [68] | 62.0 | 74.9 | 75.0 | 3.20 | 10.8 | **1,812** | **1,824** | 62.1 | 65.8 |
| QDTrack [50] | 53.9 | 68.7 | 66.3 | 2.66 | 14.7 | 3,378 | 8,091 | 52.7 | 57.2 |
| MOTR [77] | 57.2 | 71.9 | 68.4 | 2.11 | 13.6 | 2,115 | 3,897 | 55.8 | 59.2 |
| PermaTr [65] | 55.5 | 73.8 | 68.9 | 2.90 | 11.5 | 3,699 | 6,132 | 53.1 | 59.8 |
| TransMOT [16] | 61.7 | 76.7 | 75.1 | 3.62 | 9.32 | 2,346 | 7,719 | 59.9 | 66.5 |
| GTR [84] | 59.1 | 75.3 | 71.5 | 2.68 | 11.0 | 2,859 | - | 61.6 | - |
| DST-Tracker [12] | 60.1 | 75.2 | 72.3 | 2.42 | 11.0 | 2,729 | - | 62.1 | - |
| MeMOT [7] | 56.9 | 72.5 | 69.0 | 2.72 | 11.5 | 2,724 | - | 55.2 | - |
| UniCorn [76] | 61.7 | 77.2 | 75.5 | 5.01 | **7.33** | 5,379 | - | - | - |
| ByteTrack [79] | 63.1 | **80.3** | 77.3 | 2.55 | 8.37 | 2,196 | 2,277 | 62.0 | **68.2** |
| OC-SORT | **63.2** | 78.0 | **77.5** | **1.51** | 10.8 | 1,950 | 2,040 | **63.2** | 67.5 |

Table 4.2: Results on MOT20-test with private detections. ByteTrack and OC-SORT share detections.

| Tracker | HOTA↑ | MOTA↑ | IDF1↑ | FP($10^4$)↓ | FN($10^4$)↓ | IDs↓ | Frag↓ | AssA↑ | AssR↑ |
|---|---|---|---|---|---|---|---|---|---|
| FairMOT [80] | 54.6 | 61.8 | 67.3 | 10.3 | 8.89 | 5,243 | 7,874 | 54.7 | 60.7 |
| TransCt [75] | 43.5 | 58.5 | 49.6 | 6.42 | 14.6 | 4,695 | 9,581 | 37.0 | 45.1 |
| Semi-TCL [42] | 55.3 | 65.2 | 70.1 | 6.12 | 11.5 | 4,139 | 8,508 | 56.3 | 60.9 |
| CSTrack [43] | 54.0 | 66.6 | 68.6 | 2.54 | 14.4 | 3,196 | 7,632 | 54.0 | 57.6 |
| GSDT [69] | 53.6 | 67.1 | 67.5 | 3.19 | 13.5 | 3,131 | 9,875 | 52.7 | 58.5 |
| TransMOT [16] | 61.9 | 77.5 | 75.2 | 3.42 | **8.08** | 1,615 | 2,421 | 60.1 | 66.3 |
| MeMOT [7] | 54.1 | 63.7 | 66.1 | 4.79 | 13.8 | 1,938 | - | 55.0 | - |
| ByteTrack [79] | 61.3 | **77.8** | 75.2 | 2.62 | 8.76 | 1,223 | 1,460 | 59.6 | 66.2 |
| OC-SORT | **62.1** | 75.5 | **75.9** | **1.80** | 10.8 | **913** | **1,198** | **62.0** | **67.5** |

Table 4.3: Results on DanceTrack test set. Methods in blue share the same detections.

| Tracker | HOTA↑ | DetA↑ | AssA↑ | MOTA↑ | IDF1↑ |
|---|---|---|---|---|---|
| CenterTrack [83] | 41.8 | 78.1 | 22.6 | 86.8 | 35.7 |
| FairMOT [80] | 39.7 | 66.7 | 23.8 | 82.2 | 40.8 |
| QDTrack [50] | 45.7 | 72.1 | 29.2 | 83.0 | 44.8 |
| TransTrk[62] | 45.5 | 75.9 | 27.5 | 88.4 | 45.2 |
| TraDes [72] | 43.3 | 74.5 | 25.4 | 86.2 | 41.2 |
| MOTR [77] | 54.2 | 73.5 | 40.2 | 79.7 | 51.5 |
| GTR [84] | 48.0 | 72.5 | 31.9 | 84.7 | 50.3 |
| DST-Tracker [12] | 51.9 | 72.3 | 34.6 | 84.9 | 51.0 |
| SORT [5] | 47.9 | 72.0 | 31.2 | 91.8 | 50.8 |
| DeepSORT [71] | 45.6 | 71.0 | 29.7 | 87.8 | 47.9 |
| ByteTrack [79] | 47.3 | 71.6 | 31.4 | 89.5 | 52.5 |
| OC-SORT | 54.6 | **80.4** | 40.2 | 89.6 | 54.6 |
| OC-SORT + Linear Interp | **55.1** | **80.4** | **40.4** | **92.2** | **54.9** |

Table 4.4: Results on DanceTrack test set. "Ours (MOT17)" uses the YOLOX detector trained on MOT17-training set.

| Tracker | HOTA↑ | DetA↑ | AssA↑ | MOTA↑ | IDF1↑ |
|---|---|---|---|---|---|
| SORT | 47.9 | 72.0 | 31.2 | 91.8 | 50.8 |
| OC-SORT | 55.1 | 80.3 | 38.0 | 89.4 | 54.2 |
| OC-SORT (MOT17) | 48.6 | 71.0 | 33.3 | 84.2 | 51.5 |

Table 4.5: Results on KITTI-test. Our method uses the same detections as PermaTr [65]

| | Car | | | | | Pedestrian | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Tracker | HOTA↑ | MOTA↑ | AssA↑ | IDs↓ | Frag↓ | HOTA↑ | MOTA↑ | AssA↑ | IDs↓ | Frag↓ |
| IMMDP [73] | 68.66 | 82.75 | 69.76 | 211 | 181 | - | - | - | - | - |
| SMAT [27] | 71.88 | 83.64 | 72.13 | **198** | 294 | - | - | - | - | - |
| TrackMPNN [53] | 72.30 | 87.33 | 70.63 | 481 | 237 | 39.40 | 52.10 | 35.45 | 626 | 669 |
| MPNTrack [6] | - | - | - | - | - | 45.26 | 46.23 | 47.28 | 397 | 1,078 |
| CenterTr [83] | 73.02 | 88.83 | 71.18 | 254 | 227 | 40.35 | 53.84 | 36.93 | 425 | 618 |
| LGM [67] | 73.14 | 87.60 | 72.31 | 448 | **164** | - | - | - | - | - |
| TuSimple [15] | 71.55 | 86.31 | 71.11 | 292 | 218 | 45.88 | 57.61 | 47.62 | 246 | 651 |
| PermaTr [65] | **77.42** | **90.85** | **77.66** | 275 | 271 | 47.43 | 65.05 | 43.66 | 483 | 703 |
| OC-SORT | 74.64 | 87.81 | 74.52 | 257 | 318 | 52.95 | 62.00 | 57.81 | **181** | **598** |
| OC-SORT + HP | 76.54 | 90.28 | 76.39 | 250 | 280 | **54.69** | **65.14** | **59.08** | 184 | 609 |

## 4.2 Benchmark Results

Here we report the benchmark results on multiple datasets. We put all methods that use the shared detection results in the blue blocks at the bottom of each table.

Table 4.6: Results on CroHD Head Tracking dataset [64]. Our method uses the detections from HeadHunter [64] or FairMOT [80] to generate new tracks.

| Tracker | HOTA↑ | MOTA↑ | IDF1↑ | FP($10^4$)↓ | FN($10^4$)↓ | IDs↓ | Frag↓ |
|---|---|---|---|---|---|---|---|
| HeadHunter [64] | 36.8 | 57.8 | 53.9 | **5.18** | 30.0 | 4,394 | 15,146 |
| HeadHunter dets + OC-SORT | 39.0 | 60.0 | 56.8 | **5.18** | 28.1 | **4,122** | 10,483 |
| FairMOT [80] | 43.0 | 60.8 | 62.8 | 11.8 | 19.9 | 12,781 | 41,399 |
| FairMOT dets + OC-SORT | **44.1** | **67.9** | **62.9** | 10.2 | **16.4** | 4,243 | **10,122** |

### 4.2.1 Results on MOT17 and MOT20

**Private tracking.** We report OC-SORT's performance on MOT17 and MOT20 in Table 4.1 and Table 4.2 using private detections. To make a fair comparison, we use the same detection as ByteTrack [79]. OC-SORT achieves performance comparable to other state-of-the-art methods. Our gains are especially significant in MOT20 under severe pedestrian occlusion, setting a state-of-the-art HOTA of 62.1. As our method is designed to be simple for better generalization, we do not use adaptive detection thresholds as in ByteTrack. Also, ByteTrack uses more detections of low-confidence to achieve higher MOTA scores but we keep the

Table 4.7: Results on MOT17 test set with the public detections. LI indicates Linear Interpolation.

| Tracker | HOTA↑ | MOTA↑ | IDF1↑ | FP($10^4$)↓ | FN($10^4$)↓ | IDs↓ | Frag↓ | AssA↑ | AssR↑ |
|---|---|---|---|---|---|---|---|---|---|
| CenterTrack [83] | - | 61.5 | 59.6 | 1.41 | 20.1 | 2,583 | - | - | - |
| QDTrack [50] | - | 64.6 | 65.1 | 1.41 | 18.3 | 2,652 | - | - | - |
| Lif_T [32] | 51.3 | 60.5 | 65.6 | 1.50 | 20.7 | 1,189 | 3,476 | 54.7 | 59.0 |
| TransCt [75] | 51.4 | 68.8 | 61.4 | 2.29 | **14.9** | 4,102 | 8,468 | 47.7 | 52.8 |
| TrackFormer [47] | - | **62.5** | 60.7 | 3.28 | 17.5 | 2,540 | - | - | - |
| OC-SORT | 52.4 | 58.2 | 65.1 | **0.44** | 23.0 | **784** | 2,006 | **57.6** | 63.5 |
| OC-SORT + LI | **52.9** | 59.4 | 65.7 | 0.66 | 22.2 | 801 | **1,030** | 57.5 | **63.9** |

Table 4.8: Results on MOT20 test set with the public detections. LI indicates Linear Interpolation.

| Tracker | HOTA↑ | MOTA↑ | IDF1↑ | FP($10^4$)↓ | FN($10^4$)↓ | IDs↓ | Frag↓ | AssA↑ | AssR↑ |
|---|---|---|---|---|---|---|---|---|---|
| MPNTrack [6] | 46.8 | 57.6 | 59.1 | 17.0 | 20.1 | 1,210 | 1,420 | 47.3 | 52.7 |
| TransCt [75] | 43.5 | 61.0 | 49.8 | 4.92 | **14.8** | 4,493 | 8,950 | 36.1 | 44.5 |
| ApLift [33] | 46.6 | 58.9 | 56.5 | 1.77 | 19.3 | 2,241 | 2,112 | 45.2 | 48.1 |
| TMOH [61] | 48.9 | 60.1 | 61.2 | 3.80 | 16.6 | 2,342 | 4,320 | 48.4 | 52.9 |
| LPC_MOT [18] | 49.0 | 56.3 | 62.5 | 1.17 | 21.3 | 1,562 | 1,865 | 52.4 | 54.7 |
| OC-SORT | 54.3 | 59.9 | 67.0 | **0.44** | 20.2 | 554 | 2,345 | 59.5 | 65.1 |
| OC-SORT + LI | **55.2** | **61.7** | **67.9** | 0.57 | 19.2 | **508** | **805** | **59.8** | **65.9** |

Table 4.9: Influence from the value of $\Delta t$ in OCM.

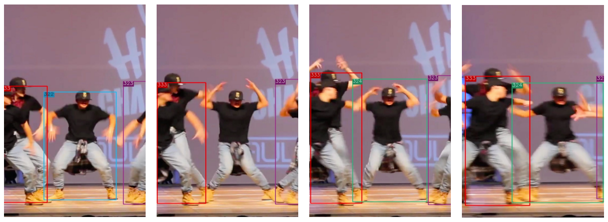
| | MOT17-val | | | DanceTrack-val | | |
|---|---|---|---|---|---|---|
| | HOTA↑ | AssA↑ | IDF1↑ | HOTA↑ | AssA↑ | IDF1↑ |
| $\Delta t = 1$ | 66.1 | 67.5 | 76.9 | 51.3 | 34.3 | 51.3 |
| $\Delta t = 2$ | 66.3 | 68.0 | 77.3 | **52.2** | **35.4** | 51.4 |
| $\Delta t = 3$ | **66.5** | **68.9** | **77.7** | 52.1 | 35.3 | 51.6 |
| $\Delta t = 6$ | 66.0 | 67.5 | 76.9 | 52.1 | **35.4** | **51.8** |

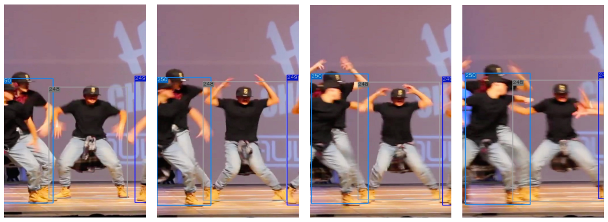
detection confidence threshold the same as on other datasets, which is the common practice in the community. We inherit the linear interpolation on the two datasets by baseline methods for a fair comparison.

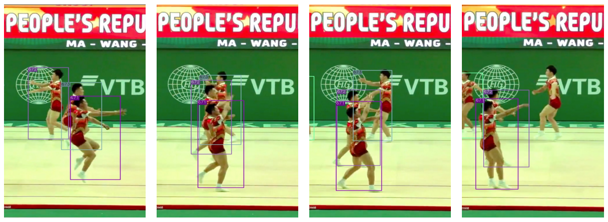
**Public tracking.** Although we use the same object detectors as some selected baselines, there are still variances in detections when compared with other methods. Therefore, we also report the public detections on MOT17/MOT20 in Table 4.7 and Table 4.8. OC-SORT still outperforms the existing state-of-the-arts in the public tracking setting. And the outperforming of OC-SORT is more significant on MOT20 which has more severe occlusion scenes. Some samples from the test set of MOT20 are shown in the last row in Figure 5.2.

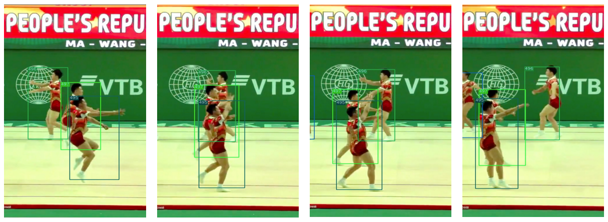
### 4.2.2 Results on DanceTrack

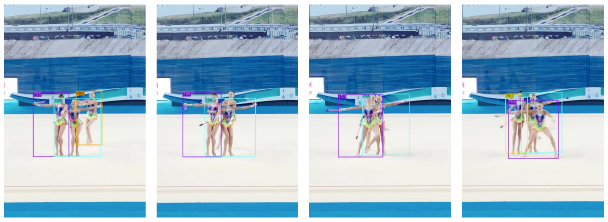
To evaluate OC-SORT under challenging non-linear object motion, we report results on the DanceTrack in Table 4.3. OC-SORT sets a new state-of-the-art, outperforming the baselines

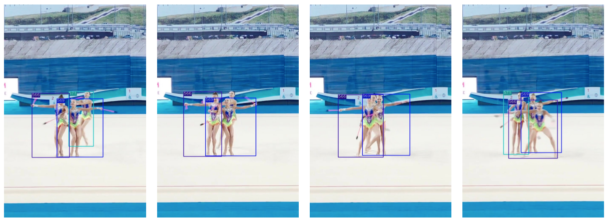

(a) SORT: dancetrack0036

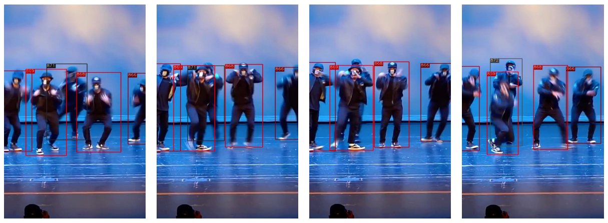

(b) OC-SORT: dancetrack0036

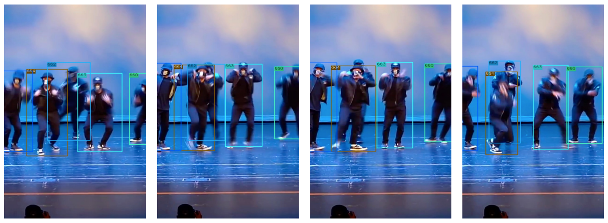

(c) SORT: dancetrack0054

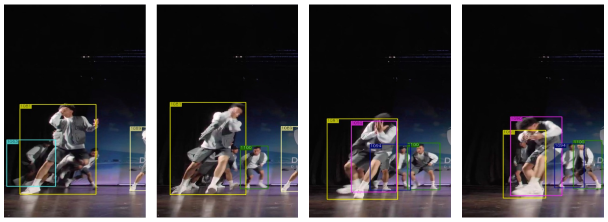

(d) OC-SORT: dancetrack0054

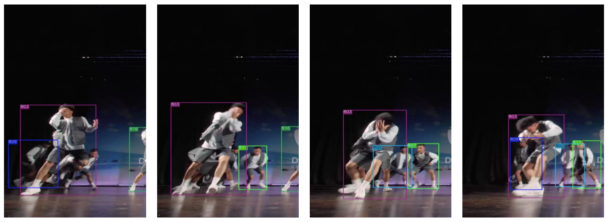

(e) SORT: dancetrack0064

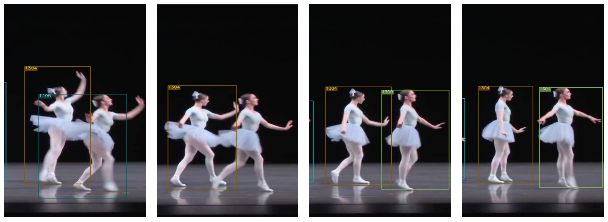

(f) OC-SORT: dancetrack0064


(g) SORT: dancetrack0078


(h) OC-SORT: dancetrack0078


(i) SORT: dancetrack0089


(j) OC-SORT: dancetrack0089


(k) SORT: dancetrack0100

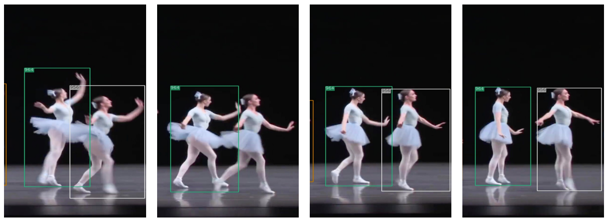

(l) OC-SORT: dancetrack0100

Figure 4.1: More samples where SORT suffers from the fragmentation and ID switch of tracks from occlusion or non-linear motion but OC-SORT survives. We selected samples from diverse scenes. Best viewed in color and zoomed in.

by a great margin under non-linear object motions. We compare the tracking results of SORT and OC-SORT under extreme non-linear situations in Figure 1.1 and more samples are available in Figure 4.1. We also visualize the output trajectories by OC-SORT and SORT on randomly selected DanceTrack video clips in Figure 4.2. For multi-object tracking in occlusion and non-linear motion, the results on DanceTrack are strong evidence of the effectiveness of OC-SORT.

To gain more intuition about the improvement of OC-SORT over SORT, we provide more comparisons. In Figure 4.1, we show more samples where SORT suffers from ID switch or Fragmentation caused by non-linear motion or occlusion but OC-SORT survives. Furthermore, in Figure 4.2, we show more samples of trajectory visualizations from SORT and OC-SORT on DanceTrack-val set.

DanceTrack [63] is proposed to encourage better association algorithms instead of carefully tuning detectors. We train YOLOX [25] detector on MOT17 training set only to provide detections on DanceTrack. We find the tracking performance of OC-SORT is already higher than the baselines (Table 4.4). We believe the potential to improve multi-object tracking by better association strategy is still promising and DanceTrack is a good platform for the evaluation.

### 4.2.3 Results on KITTI

In Table 4.5 we report the results on the KITTI dataset. For a fair comparison, we adopt the detector weights by PermaTr [65] and report its performance in the table as well. We run OC-SORT given the shared detections. As initializing SORT's track requires continuous tracking across several frames ("minimum hits"), we observe that the results not recorded during the track initialization make a significant difference. To address this problem, we perform offline head padding (HP) post-processing by writing these entries back after finishing the online tracking stage. The results of the car category on KITTI show an essential shortcoming of the default implementation version of OC-SORT that it chooses the IoU matching for the association. When the objects move fast or the frame rate is low, the IoU of bounding boxes between consecutive frames can be very low or even zero. This issue does not come from the intrinsic design of OC-SORT and is widely observed when using IoU as the association cue. Adding other cues [57, 81, 83] and appearance similarity [46, 71] have been demonstrated [71] effective to solve this. In contrast to the relatively inferior car tracking performance, OC-SORT improves pedestrian tracking performance to a new state-of-the-art. Using the same detections, OC-SORT achieves a large performance gap over PermaTr with 10x faster speed.

### 4.2.4 Results on HeadTrack

When considering tracking in the crowd, focusing on only a part of the object can be beneficial [8] as it usually suffers less from occlusion than the full body. This line of study is conducted over hand tracking [48, 58], human pose [74] and head tracking [3, 51, 64] for a while. Moreover, with the knowledge of more fine-grained part trajectory, it can be useful in downstream tasks, such as action recognition [22, 23] and forecasting [9, 13, 38, 40]. As we are interested in the multi-object tracking in the crowd, we also evaluate the proposed OC-SORT on a recently proposed human head tracking dataset CroHD [64]. To make a fair comparison on only the association performance, we adopt OC-SORT by directing using the detections from existing tracking algorithms. The results are shown in Table 4.6. The detections of FairMOT [80] and HeadHunter [64] are extracted from their tracking results downloaded from the official leaderboard [2]. We use the same parameters for OC-SORT as on the other datasets. The results suggest a significant tracking performance improvement compared with the previous methods [64, 80] for human body part tracking. But the tracking performance is still relatively low (HOTA=$\sim$ 40). It is highly related to the difficulty of having accurate detection of tiny objects. Some samples from the test set of HeadTrack are shown in the first two rows of Figure 5.2.

The results on multiple benchmarks have demonstrated the effectiveness and efficiency of OC-SORT. We note that we use a shared parameter stack across datasets. Carefully tuning the parameters can probably further boost the performance. For example, the adaptive detection threshold is proven useful in previous work [79]. Besides the association accuracy, we also care about the inference speed. Given off-the-shelf detections, OC-SORT runs at 793 FPS on an Intel i9-9980XE CPU @ 3.00GHz. Therefore, OC-SORT can still run in an online and real-time fashion.

Table 4.10: Ablation on MOT17-val and DanceTrack-val.

| | | | MOT17-val | | | DanceTrack-val | | |
|---|---|---|---|---|---|---|---|---|
| ORU | OCM | OCR | HOTA↑ | AssA↑ | IDF1↑ | HOTA↑ | AssA↑ | IDF1↑ |
| | | | 64.9 | 66.8 | 76.9 | 47.8 | 31.0 | 48.3 |
| ✓ | | | 66.3 | 68.0 | 77.2 | 48.5 | 32.2 | 49.8 |
| ✓ | ✓ | | 66.4 | **69.0** | **77.8** | **52.1** | 35.0 | 50.6 |
| ✓ | ✓ | ✓ | **66.5** | 68.9 | 77.7 | **52.1** | **35.3** | **51.6** |

[2]https://motchallenge.net/results/Head_Tracking_21/

Table 4.11: Ablation on the trajectory hypothesis in ORU.

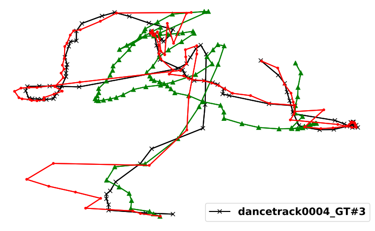
| | MOT17-val | | | DanceTrack-val | | |
|---|---|---|---|---|---|---|
| | HOTA↑ | AssA↑ | IDF1↑ | HOTA↑ | AssA↑ | IDF1↑ |
| Const. Speed | **66.5** | **68.9** | **77.7** | **52.1** | **35.3** | **51.6** |
| GPR | 63.1 | 65.2 | 75.7 | 49.5 | 33.7 | 49.6 |
| Linear Regression | 64.3 | 66.5 | 76.0 | 49.3 | 33.4 | 49.2 |
| Const. Acceleration | 66.2 | 67.9 | 77.4 | 51.3 | 34.8 | 50.9 |

## 4.3 Ablation Study

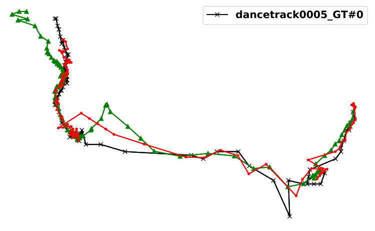
**Component Ablation.** We ablate the contribution of proposed modules on the validation sets of MOT17 and DanceTrack in Table 4.10. The splitting of the MOT17 validation set follows a popular convention [83]. The results demonstrate the effectiveness of the proposed modules in OC-SORT. The results show that the performance gain from ORU is significant on both datasets but OCM only shows significant help on DanceTrack dataset where object motion is more complicated and the occlusion is heavy. The ablation study proves the effectiveness of our proposed method to improve tracking robustness in occlusion and non-linear motion.

**Virtual Trajectory in ORU.** For simplicity, we follow the naive hypothesis of constant speed to generate a virtual trajectory in ORU. There are other alternatives like constant acceleration, regression-based fitting such as Linear Regression (LR) or Gaussian Process Regression (GPR), and Near Constant Acceleration Model (NCAM) [34]. The results of comparing these choices are shown in Table 4.11. For GPR, we use the RBF kernel [14] $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{50}\right)$. We provide more studies on the kernel configuration in Section 5.2.1. The results show that local hypotheses such as Constant Speed/Acceleration perform much better than global hypotheses such as LR and GPR. This is probably because, as virtual trajectory generation happens in an online fashion, it is hard to get a reliable fit using only limited data points on historical time steps.

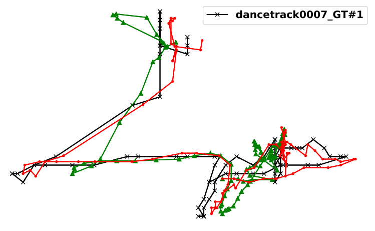
**$\Delta t$ in OCM.** There is a trade-off when choosing the time difference $\Delta t$ in OCM (Section 3.2). A large $\Delta t$ decreases the noise of velocity estimation. but is also likely to discourage approximating object motion as linear. Therefore, we study the influence of varying $\Delta t$ in Table 4.9. Our results agree with our analysis that increasing $\Delta t$ from $\Delta t = 1$ can boost performance. But increasing $\Delta t$ higher than a best-practice value instead hurts the performance because of the difficulty of maintaining the approximation of linear motion.



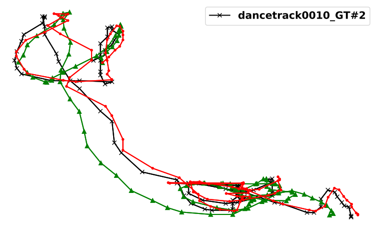
(a) GT #3 on video #0003



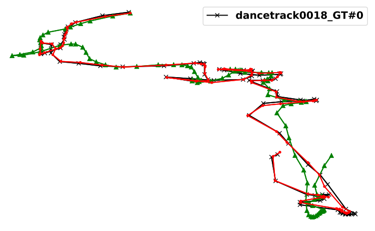
(b) GT #0 on video #0005



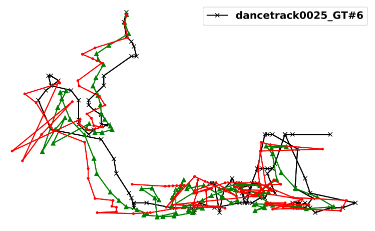
(c) GT #1 on video #0007



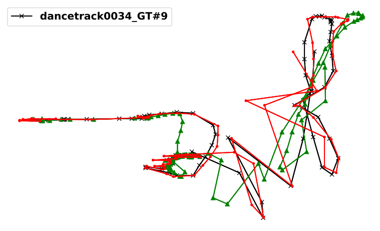
(d) GT #2 on video #0010



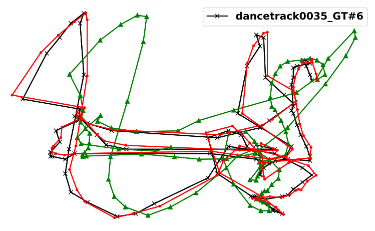
(e) GT #0 on video #0018



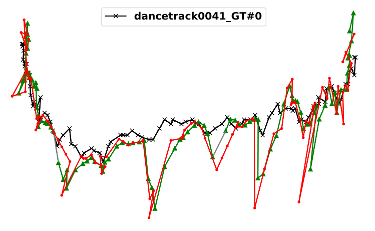
(f) GT #6 on video #0025



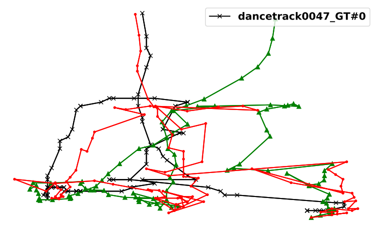
(g) GT #9 on video #0034



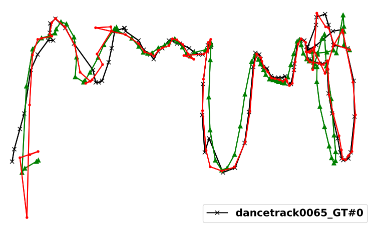
(h) GT #6 on video #0035



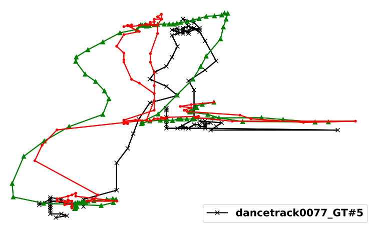
(i) GT #0 on video #0041



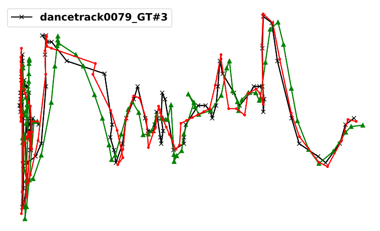
(j) GT #0 on video #0047



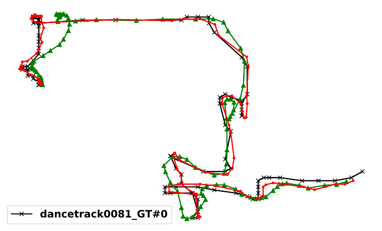
(k) GT #0 on video #0065



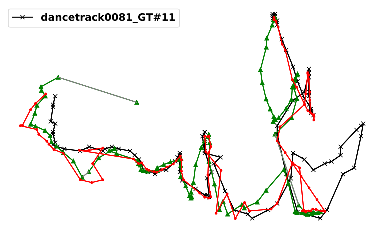
(l) GT #5 on video #0077



(m) GT #3 on video #0079



(n) GT #0 on video #0081



(o) GT #11 on video #0081

Figure 4.2: Trajectory samples from Dancetrack-val set. **black cross**: ground truth trajectory; **red dots**: trajectory output by OC-SORT **green triangles**: trajectory output by SORT. SORT and OC-SORT use the same hyperparameters and detections.

# Chapter 5

# More Analysis

## 5.1 Velocity Direction Variance in OCM

In this section, we work on the setting of linear motion with noisy states. We provide proof that the trajectory direction estimation has a smaller variance if the two states we use for the estimation have a larger time difference. We assume the motion model is $\mathbf{x}_t = f(t) + \epsilon$ where $\epsilon$ is gaussian noise and the ground-truth center position of the target is $(\mu_{u_t}, \mu_{v_t})$ at time step $t$. Then the true motion direction between the two time steps is

$$\theta = \arctan(\frac{\mu_{v_{t_1}} - \mu_{v_{t_2}}}{\mu_{u_{t_1}} - \mu_{u_{t_2}}}). \tag{5.1}$$

And we have $|\mu_{v_{t_1}} - \mu_{v_{t_2}}| \propto |t_1 - t_2|$, $|\mu_{u_{t_1}} - \mu_{u_{t_2}}| \propto |t_1 - t_2|$. As the detection results do not suffer from the error accumulation due to propagating along Markov process as Kalman filter does, we can assume the states from observation suffers some i.i.d. noise, i.e., $u_t \sim \mathcal{N}(\mu_{u_t}, \sigma_u^2)$ and $v_t \sim \mathcal{N}(\mu_{v_t}, \sigma_v^2)$. We now analyze the noise of the estimated $\tilde{\theta} = \frac{v_{t_1} - v_{t_2}}{u_{t_1} - u_{t_2}}$ by two observations on the trajectory. Because the function of $\arctan(\cdot)$ is monotone over the whole real field, we can study $\tan\tilde{\theta}$ instead which simplifies the analysis. We denote $w = u_{t_1} - u_{t_2}$, $y = v_{t_1} - v_{t_2}$, and $z = \frac{y}{w}$, first we can see that $y$ and $w$ jointly form a Gaussian distribution:

$$\begin{bmatrix} y \\ w \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} \mu_y \\ \mu_w \end{bmatrix}, \begin{bmatrix} \sigma_y^2 & \rho\sigma_y\sigma_w \\ \rho\sigma_y\sigma_w & \sigma_w^2 \end{bmatrix} \right), \tag{5.2}$$

where $\mu_y = \mu_{v_{t_1}} - \mu_{v_{t2}}$, $\mu_w = \mu_{u_{t_1}} - \mu_{u_{t_2}}$, $\sigma_w = \sqrt{2}\sigma_u$ and $\sigma_y = \sqrt{2}\sigma_v$, and $\rho$ is the correlation coefficient between $y$ and $w$. We can derive a closed-form solution of the probability density function [31] of $z$ as

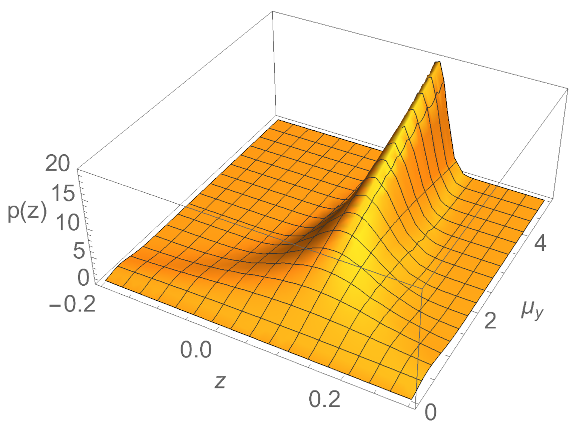
$$\begin{aligned} p(z) =& \frac{g(z)e^{\frac{g(z)^2-\alpha r(z)^2}{2\beta^2 r(z)^2}}}{\sqrt{2\pi}\sigma_w\sigma_y r(z)^3}\left[\Phi\left(\frac{g(z)}{\beta r(z)}\right)-\Phi\left(-\frac{g(z)}{\beta r(z)}\right)\right] \\ &+\frac{\beta e^{-2\alpha/\beta}}{\pi\sigma_w\sigma_y r(z)^2} \end{aligned} \tag{5.3}$$

where

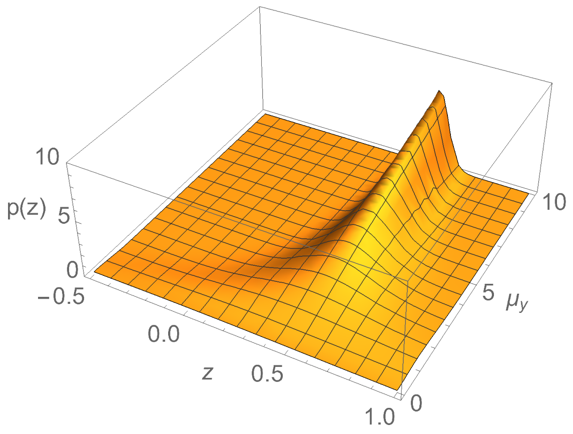
$$\begin{aligned} r(z) &= \sqrt{\frac{z^2}{\sigma_y^2}-\frac{2\rho z}{\sigma_y\sigma_w}+\frac{1}{\sigma_w^2}}, \\ g(z) &= \frac{\mu_y z}{\sigma_y^2}-\frac{\rho(\mu_y+\mu_w z)}{\sigma_y\sigma_w}+\frac{\mu_w}{\sigma_w^2}, \\ \alpha &= \frac{\mu_w^2+\mu_y^2}{\sigma_y^2}-\frac{2\rho\mu_y\mu_w}{\sigma_w\sigma_y}, \qquad \beta=\sqrt{1-\rho^2}, \end{aligned} \tag{5.4}$$

and $\Phi$ is the cumulative distribution function of the standard normal. Without loss of generality, we can assume $\mu_w > 0$ and $\mu_y > 0$ because negative ground-truth displacements enjoy the same property. This solution has a good property that larger $\mu_w$ or $\mu_y$ makes the probability density at the true value, i.e. $\mu_z = \frac{\mu_y}{\mu_w}$, higher, and the tails decay more rapidly. So the estimation of $\arctan\theta$, also $\theta$, has smaller noise when $\mu_w$ or $\mu_y$ is larger. Under the assumption of linear motion, we thus should select two observations with a large temporal difference to estimate the direction.

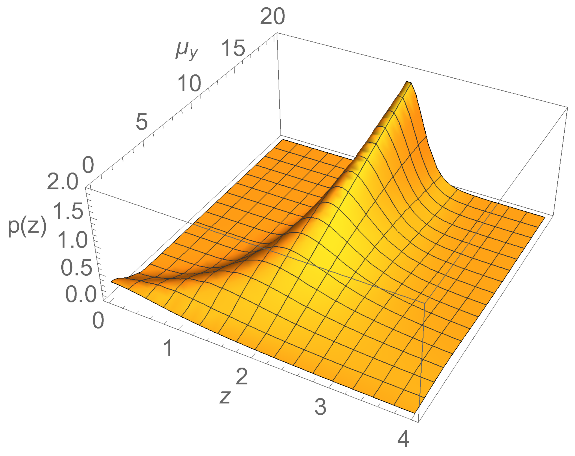
It is reasonable to assume the noise of detection along the u-axis and v-axis are independent so $\rho = 0$. And when representing the center position in pixel, it is also moderate to assume $\sigma_w = \sigma_y = 1$ (also for the ease of presentation). Then, with different true value of $\mu_z = \frac{\mu_y}{\mu_w}$, the visualizations of $p(z)$ over $z$ and $\mu_y$ are shown in Figure 5.1. The visualization demonstrates our analysis above. Moreover, it shows that when the value of $\mu_y$ or $\mu_w$ is small, the cluster peak of the distribution at $\mu_z$ is not significant anymore, as the noise $\sigma_y$ and $\sigma_w$ can be dominant. Considering the visualization shows that happens when $\mu_y$ is close to $\sigma_y$, this can happen when we estimate the speed by observations from two consecutive frames because the variance of observation can be close to the absolute displacement of object motion. This makes another support to our analysis in the main paper about the sensitivity to state estimation noise.



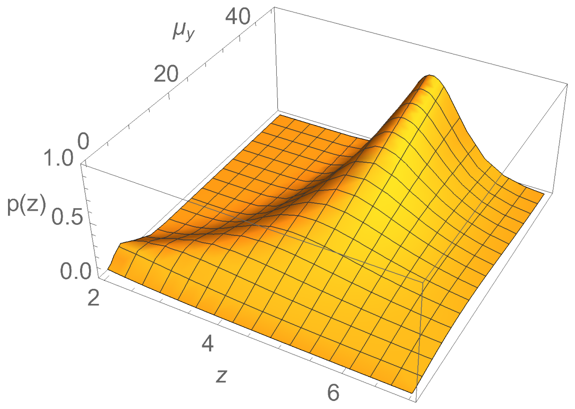
(a) $\mu_z = 0.1$



(b) $\mu_z = 0.5$



(c) $\mu_z = 2$



(d) $\mu_z = 5$

Figure 5.1: The probability density of $z = \tan\theta$ under different true value of $z$, i.e. $\mu_z = \frac{\mu_y}{\mu_w}$. We set $\mu_y$ and $z$ as two variables. It shows that under different settings of true velocity direction when $\mu_y$ is smaller, the probability of estimated value with a significant shift from the true value is higher. As $\mu_y$ is proportional to the time difference of the two selected observations under linear motion assumption, it relates to the case that the two steps for velocity direction estimation has a shorter time difference.

Table 5.1: Ablation study about the interpolation post-processing.

| | MOT17-val | | | | DanceTrack-val | | | |
|---|---|---|---|---|---|---|---|---|
| | HOTA↑ | AssA↑ | MOTA↑ | IDF1↑ | HOTA↑ | AssA↑ | MOTA↑ | IDF1↑ |
| w/o interpolation | 66.5 | 68.9 | 74.9 | 77.7 | 52.1 | 35.3 | 87.3 | 51.6 |
| Linear Interpolation | **68.0** | **69.9** | **77.9** | **79.3** | **52.8** | **35.6** | **89.8** | **52.1** |
| GPR Interpolation | 65.2 | 67.0 | 72.9 | 75.9 | 51.6 | 35.0 | 86.1 | 51.2 |

Table 5.2: Ablation study about using Gaussian Process Regression for object trajectory interpolation. LI indicates Linear Interpolation, which is used to interpolate the trajectory before smoothing the trajectory by GPR. MT indicates Median Trick for kernel choice in regression. $L_\tau$ is the length of trajectory.

| | MOT17-val | | | | DanceTrack-val | | | |
|---|---|---|---|---|---|---|---|---|
| Interpolation Method | HOTA | AssA | MOTA | IDF1 | HOTA | AssA | MOTA | IDF1 |
| w/o interpolation | 66.5 | 68.9 | 74.9 | 77.7 | 52.1 | 35.3 | 87.3 | 51.6 |
| Linear Interpolation | **69.6** | **69.9** | **77.9** | **79.3** | 52.8 | **35.6** | 89.8 | **52.1** |
| GPR Interp, $l = 1$ | 66.2 | 67.6 | 74.3 | 76.6 | 51.8 | 35.0 | 86.6 | 50.8 |
| GPR Interp, $l = 5$ | 66.3 | 67.0 | 72.9 | 75.9 | 51.8 | 35.1 | 86.5 | 51.1 |
| GPR Interp, $l = L_\tau$ | 66.1 | 67.0 | 73.1 | 77.8 | 51.6 | 35.1 | 86.4 | 50.7 |
| GPR Interp, $l = 1000/L_\tau$ | 65.9 | 67.0 | 73.0 | 77.8 | 51.8 | 35.0 | 86.9 | 51.0 |
| GPR Interp, $l = \mathrm{MT}(\tau)$ | 65.9 | 67.0 | 73.1 | 77.8 | 51.7 | 35.1 | 86.7 | 50.9 |
| LI + GPR Smoothing, $l = 1$ | 69.5 | 69.6 | 77.8 | **79.3** | 52.8 | **35.6** | **89.9** | **52.1** |
| LI + GPR Smoothing, $l = 5$ | 69.5 | 69.7 | 77.8 | **79.3** | 52.9 | 34.9 | 89.7 | **52.1** |
| LI + GPR Smoothing, $l = L_\tau$ | 69.6 | 69.5 | 77.8 | 79.2 | 52.9 | **35.6** | **89.9** | **52.1** |
| LI + GPR Smoothing, $l = 1000/L_\tau$ | 69.5 | **69.9** | 77.8 | **79.3** | **53.0** | **35.6** | **89.9** | **52.1** |
| LI + GPR Smoothing, $l = \mathrm{MT}(\tau)$ | 69.5 | 69.6 | 77.8 | **79.3** | 52.8 | **35.6** | 89.8 | **52.1** |

## 5.2 More Experiments

### 5.2.1 Interpolation by Gaussian Progress Regression

**Interpolation as post-processing.** Although we focus on developing an online tracking algorithm, we are also interested in whether post-process can further optimize the tracking results in diverse conditions. Despite the failure of GPR in online tracking in Table 4.11, we continue to study if GPR is better suited for interpolation in Table 5.1. We compare GPR with the widely-used linear interpolation. The maximum gap for interpolation is set as 20 frames and we use the same kernel for GPR as mentioned above. The results suggest that the GPR's non-linear interpolation is simply not efficient. We think this is due to limited data points which results in an inaccurate fit of the object trajectory. Further, the variance in regressor predictions introduces extra noise. Although GPR interpolation decreases the performance on MOT17-val significantly, its negative influence on DanceTrack is relatively minor where the object motion is more non-linear. We believe how to fit object trajectory with non-linear hypothesis still requires more study.

From the analysis in the main paper, the failure of SORT can mainly result from occlusion (lack of observations) or the non-linear motion of objects (the break of the linear-motion assumption). So the question arises naturally whether we can extend SORT free of the linear-motion assumption or at least more robust when it breaks.

One way is to extend from KF to non-linear filters, such as EKF [37, 60] and UKF [35]. However, for real-world online tracking, they can be hard to be adopted as they need knowledge about the motion pattern or still rely on the techniques fragile to non-linear patterns, such as linearization [36]. Another choice is to gain the knowledge beyond linearity by regressing previous trajectory, such as combing Gaussian Process (GP) [39, 55, 70]: given a observation $\mathbf{z}_\star$ and a kernel function $k(\cdot,\cdot)$, GP defines gaussian functions with mean $\mu_{\mathbf{z}_\star}$ and variance $\Sigma_{\mathbf{z}_\star}$ as

$$\begin{aligned}\mu_{\mathbf{z}_\star} &= \mathbf{k}_\star^\top[\mathbf{K}+\sigma^2\mathbf{I}]^{-1}\mathbf{y},\\ \Sigma_{\mathbf{z}_\star} &= k(\mathbf{z}_\star,\mathbf{z}_\star)-\mathbf{k}_\star^\top[\mathbf{K}+\sigma^2\mathbf{I}]^{-1}\mathbf{k}_\star,\end{aligned} \tag{5.5}$$

where $\mathbf{k}_\star$ is the kernel matrix between the input and training data and $\mathbf{K}$ is the kernel matrix over training data, $\mathbf{y}$ is the output of data. Until now, we have shown the primary study of using Gaussian Process Regression (GPR) in the online generation of the virtual trajectory in ORU and offline interpolation. But neither of them successfully boosts the tracking performance. Now, We continue to investigate in detail the chance of combining GPR and SORT for multi-object tracking for interpolation as some designs are worth more study.

### 5.2.2 Choice of Kernel Function in Gaussian Process

The kernel function is a key variable of GPR. There is not a generally efficient guideline to choose the kernel for Gaussian Process Regression though some basic observations are available [21]. When there is no additional knowledge about the time sequential data to fit, the RBF kernel is one of the most common choices:

$$k(\mathbf{x},\mathbf{x}') = \sigma^2 \exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2l^2}\right), \tag{5.6}$$

where $l$ is the lengthscale of the data to be fit. It determines the length of the “wiggles” of the target function. $\sigma^2$ is the output variance that determines the average distance of the function away from its mean. This is usually just a scale factor [21]. GPR is considered sensitive to $l$ in some situations. So we conduct an ablation study over it in the offline interpolation to see if we can use GPR to outperform the linear interpolation widely used in multi-object tracking.

### 5.2.3 GPR for Offline Interpolation

We have presented the use of GPR in online virtual trajectory fitting and offline interpolation where we use $l^2 = 25$ and $\sigma = 1$ for the kernel in Equation (5.6). Further, we make a more thorough study of the setting of GPR. We follow the settings of experiments in the main paper that only trajectories longer than 30 frames are put into interpolation. And the interpolation is only applied to the gap shorter than 20 frames. We conduct the experiments on the validation sets of MOT17 and DanceTrack.

For the value of $l$, we try fixed values, i.e. $l = 1$ and $l = 5$ ($2l^2 = 50$), value adaptive to trajectory length, i.e. $l = L_\tau$ and $l = 1000/L_\tau$, and the value output by Median Trick (MT) [24]. The training data is a series of quaternary $[u, v, w, h]$, normalized to zero-mean before being fed into training. The results are shown in Table 5.2. Linear interpolation is simple but builds a strong baseline as it can stably improve the tracking performance concerning multiple metrics. Directly using GPR to interpolate the missing points hurts the performance and the results of GPR are not sensitive to the setting of $l$.

There are two reasons preventing GPR from accurately interpolating missing segments. First, the trajectory is usually limited to at most hundreds of steps, providing very limited data points for GPR training to converge. On the other hand, the missing intermediate data points make the data series discontinuous, causing a huge challenge. We can fix the second issue by interpolating the trajectory with Linear Interpolation (LI) first and then smoothing the interpolated steps by GPR. This outperforms LI on DanceTrack but still regrades the performance by LI on MOT17. This is likely promoted by the non-linear motion on DanceTrack. By fixing the missing data issue of GPR, GPR can have a more accurate trajectory fitting over LI for the non-linear trajectory cases. But considering the outperforming from GPR is still minor compared with the Linear Interpolation-only version and GPR requires much heavier computation overhead, we do not recommend using such a practice in most multi-object tracking tasks. More careful and deeper study is still required on this problem.

## 5.3 More Discussion of State Noise Sensitivity

We have shown that the noise of state estimate will be amplified to the noise of velocity estimate. This is because the velocity estimate is correlated to the state estimate. But the analysis is in a simplified model in which velocity itself does not gain noise from the transition directly and the noise of state estimate is i.i.d on different steps. However, in the general case, such a simplification does not hold. We now provide a more general analysis of the state noise sensitivity of SORT.

For the process in Equation (3.1), we follow the most commonly adapted implementation of Kalman filter [1] and SORT [2] for video multi-object tracking. Instead of writing the mean

[1]https://github.com/rlabbe/filterpy
[2]https://github.com/abewley/sort

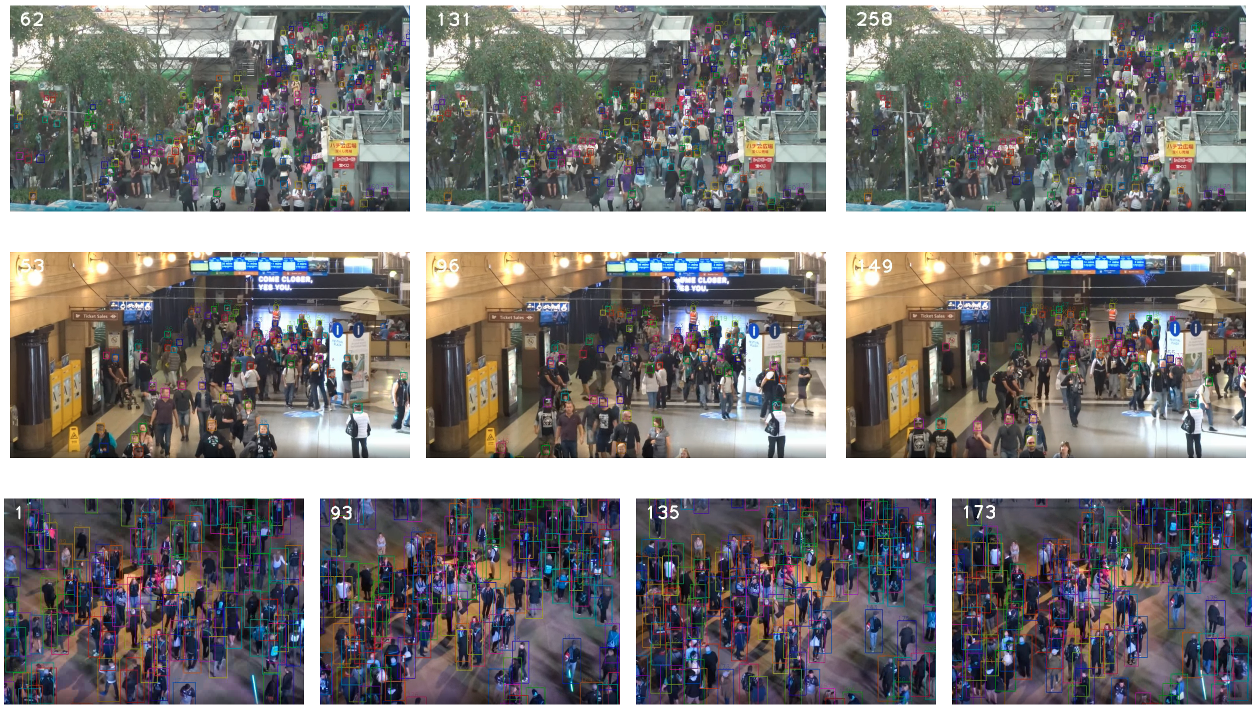


Figure 5.2: The visualization of the output of OC-SORT on randomly selected samples from the test set of HeadTrack [64] (the first two rows) and MOT20 [19] (the bottom row). These two datasets are both challenging because of the crowded scenes where pedestrians have heavy occlusion with each other. OC-SORT achieves superior performance on both datasets.

state estimate, we consider the noisy prediction of state estimate now, which is formulated as

$$\mathbf{x}_{t|t-1} = \mathbf{F}_t \mathbf{x}_{t|t-1} + \mathbf{w}_t, \tag{5.7}$$

where $\mathbf{w}_t$ is the process noise, drawn from a zero mean multivariate normal distribution, $\mathcal{N}$, with covariance, $\mathbf{w}_t \sim \mathcal{N}(0, \mathbf{Q}_t)$. As $\mathbf{x}_t$ is a seven-tuple, i.e.$\mathbf{x}_t = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^\top$, the process noise applies to not just the state estimate but also the velocity estimates. Therefore, for a general form of analysis of temporal error magnification in Equation (3.5), we would get a different result because not just the position terms but also the velocity terms gain noise from the transition process. And the noise of velocity terms will amplify the noise of position estimate by the transition at the next step. We note the process noise as in practice:

$$\mathbf{Q}_t = \begin{bmatrix} \sigma_u^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_v^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_s^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_r^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{\dot{u}}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{\dot{v}}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma_{\dot{s}}^2 \end{bmatrix}, \tag{5.8}$$

and the linear transition model as

$$\mathbf{F}_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.9}$$

We assume the time step when a track gets untracked is $t_1$ and don't consider the noise from previous steps. For simplicity, we assume the motion in the x-direction and y-direction do not correlate. We take the motion on the x-direction as an example without loss of generality:

$$\delta_{u_{t_0}} \sim \mathcal{N}(0, \sigma_u^2), \quad \delta_{\dot{u}_{t_0}} \sim \mathcal{N}(0, {\sigma_{\dot{u}}}^2). \tag{5.10}$$

On the next step, with no correction from the observation, the error would be accumulated

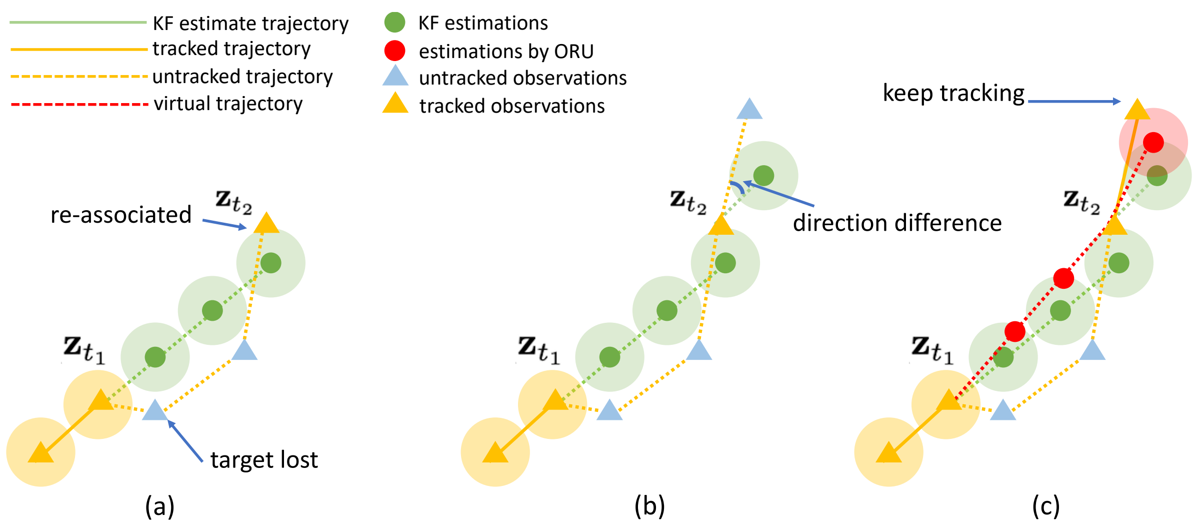


Figure 5.3: Illustration of how ORU changes the behaviors of SORT after an untracked track is re-associated to an observation. **(a).** The track is re-associates with an observation $\mathbf{z}_{t_2}$. **(b).** Without ORU, on the next step of re-association, there is still a direction difference between the true object trajectory and the KF estimates. Therefore, the track is unmatched with detections again (in blue). **(c).** With ORU, we get a more significant change in the state, especially the motion direction by updating velocity.

$(\Delta t = 1)$,

$$\delta_{u_{t_0+1}} \sim \mathcal{N}(0, 2\sigma_u^2 + \sigma_{\dot{u}}{}^2), \quad \delta_{\dot{u}_{t_0+1}} \sim \mathcal{N}(0, 2\sigma_{\dot{u}}{}^2). \tag{5.11}$$

Therefore, the accumulation is even faster than we analyze in Section 3.1.2 as

$$\delta_{u_{t_0+T}} \sim \mathcal{N}(0, (T+1)\sigma_u^2 + \frac{1}{2}T(T+1)\sigma_{\dot{u}}^2). \tag{5.12}$$

In the practice of SORT, we have to suppress the noise from velocity terms because it is too sensitive. We achieve it by setting a proper value for the process noise $\mathbf{Q}_t$. For example, the most commonly adopted value [3] of $\mathbf{Q}_t$ in SORT is

$$\mathbf{Q}_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0001 \end{bmatrix}. \tag{5.13}$$

[3]https://github.com/abewley/sort/blob/master/sort.py#L111

In such a parameter setting, we have the ratio between the noise from position terms and velocity terms as

$$\beta = \frac{(T+1)\sigma_u^2}{0.5T(T+1)\sigma_{\dot{u}}^2} = \frac{200}{T}. \tag{5.14}$$

In practice, a track is typically deleted if it keeps untracked for $T_{\text{del}}$ time steps. Usually we set $T_{\text{del}} < 10$, so we have $\beta > 20$. Therefore, we usually consider the noise from velocity terms as secondary. Such a convention allows us to use the simplified model for noise analysis. But it also brings a side-effect that SORT can't allow the velocity direction of a track to change quickly in a short time interval. We will see later (Section 5.4) that it makes trouble to SORT when non-linear motion and occlusion come together and motivates the design of ORU in OC-SORT.

## 5.4 Intuition behind ORU

ORU is designed to fix the error accumulated during occlusion when an untracked track is re-associated with an observation. But in general, the bias in the state estimate $\hat{\mathbf{x}}$ after being untracked for $T$ time steps can be fixed by the update stage once it gets re-associated with an observation. To be precise, the Optimal Kalman gain, i.e.$\mathbf{K}_t$, can use the re-associated observation to update the KF posteriori parameters. In general, such an expectation of KF's behavior is reasonable. But because we usually set the corresponding covariance for velocity terms very small (Equation (5.13)), it is difficult for SORT to steer to the correct velocity direction at the step of re-association.

Motivated by such observations, we design ORU. In the simplified model shown in Figure 5.3, the circle area with the shadow around each estimate footage is the eligible range to associate with observations inside. ORU is designed for the case that a track is re-associated after being untracked. Therefore, the typical situation is as shown in the figure that the true trajectory first goes away from the linear trajectory of KF estimates and then goes closer to it so that there can be a re-association. After the re-association, there would be a cross of the two trajectories.

In SORT, after re-associating with an observation, the direction of the velocity of the previously untracked track still has a significant difference from the true value. This is shown in Figure 5.3(b). This makes the estimate on the future steps lost again (the blue triangle). The reason is the convention of $\mathbf{Q}$ discussed in Section 5.3. Therefore, even though the canonical KF can support fixing the accumulated error during being untracked theoretically, it is very rare in practice. In ORU, we follow the virtual trajectory where we have multiple virtual observations. In this way, even if the value of $\mathbf{Q}[4:, 4:]$ is small, we can still have a better-calibrated velocity direction after the time step $t_2$. We would like to note that the intuition behind ORU is from our observations in practice and based on the common convention of using Kalman filter for multi-object tracking. It does not make fundamental changes to upgrade the power of the canonical Kalman filter.

Here we provide a more formal mathematical expression to compare the behaviors of SORT and OC-SORT. Assume that the track was lost at the time step $t_1$ and re-associated at $t_2$. We assume the mean state estimate is

$$\hat{\mathbf{x}}_{t_1|t_1} = [u_1, v_1, s_1, r_1, \dot{u}_1, \dot{v}_1, \dot{s}_1]^\top, \tag{5.15}$$

and the covariance at $t_1$ is

$$\mathbf{P}_{t_1|t_1} = \begin{bmatrix} \sigma_{u_1}^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{v_1}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{s_1}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{r_1}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{\dot{u}_1}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{\dot{v}_1}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma_{\dot{s}_1}^2 \end{bmatrix}. \tag{5.16}$$

Then, because the covariance expands from the input of process noise at each step of *predict*, at $t_2$, we have the priori estimates ($t_\Delta = t_2 - t_1$) of state

$$\hat{\mathbf{x}}_{t_2|t_2-1} = [u_2, v_2, s_2, r_2, \dot{u}_2, \dot{v}_2, \dot{s}_2]^\top, \tag{5.17}$$

with

$$\begin{aligned} u_2 &= u_1 + \dot{u}_1 t_\Delta, \\ v_2 &= v_1 + \dot{v}_1 t_\Delta, \\ s_2 &= s_1 + \dot{s}_1 t_\Delta, \\ r_2 &= r_1, \\ \dot{u}_2 &= \dot{u}_1, \\ \dot{v}_2 &= \dot{v}_1, \\ \dot{s}_2 &= \dot{s}_1. \end{aligned} \tag{5.18}$$

And the priori covariance

$$\mathbf{P}_{t_2|t_2-1} = \begin{bmatrix} \sigma^2_{u_2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{v_2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{s_2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{r_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{\dot{u}_2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma^2_{\dot{v}_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma^2_{\dot{s}_2} \end{bmatrix}, \tag{5.19}$$

with

$$\begin{aligned} \sigma^2_{u_2} &= \sigma^2_{u_1} + t_\Delta(\sigma^2_u + \sigma^2_{\dot{u}_1}) + \frac{t_\Delta(t_\Delta - 1)}{2}\sigma^2_{\dot{u}}, \\ \sigma^2_{v_2} &= \sigma^2_{v_1} + t_\Delta(\sigma^2_v + \sigma^2_{\dot{v}_1}) + \frac{t_\Delta(t_\Delta - 1)}{2}\sigma^2_{\dot{v}}, \\ \sigma^2_{s_2} &= \sigma^2_{s_1} + t_\Delta(\sigma^2_s + \sigma^2_{\dot{s}_1}) + \frac{t_\Delta(t_\Delta - 1)}{2}\sigma^2_{\dot{s}}, \\ \sigma^2_{r_2} &= \sigma^2_{r_1} + t_\Delta\sigma^2_r, \\ \sigma^2_{\dot{u}_2} &= \sigma^2_{\dot{u}_1} + t_\Delta\sigma^2_{\dot{u}}, \\ \sigma^2_{\dot{v}_2} &= \sigma^2_{\dot{v}_1} + t_\Delta\sigma^2_{\dot{v}}, \\ \sigma^2_{\dot{s}_2} &= \sigma^2_{\dot{s}_1} + t_\Delta\sigma^2_{\dot{s}}. \end{aligned} \tag{5.20}$$

Now, SORT will keep going forward as normal. Therefore, with the re-associated observation $\mathbf{z}_{t_2}$, we have

$$SORT \begin{cases} \hat{\mathbf{x}}_{t_2|t_2} = \hat{\mathbf{x}}_{t_2|t_2-1} + \mathbf{K}_{t_2}(\mathbf{z}_{t_2} - \mathbf{H}\hat{\mathbf{x}}_{t_2|t_2-1}), \\ \mathbf{P}_{t_2|t_2} = (\mathbf{I} - \mathbf{K}_{t_2}\mathbf{H})\mathbf{P}_{t_2|t_2-1} \end{cases} \tag{5.21}$$

where the observation model is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \tag{5.22}$$

and the Kalman gain is

$$\mathbf{K}_{t_2} = \mathbf{P}_{t_2|t_2-1}\mathbf{H}^\top(\mathbf{H}\mathbf{P}_{t_2|t_2-1}\mathbf{H}^\top + \mathbf{R}_{t_2})^{-1}. \tag{5.23}$$

On the other hand, OC-SORT will replay Kalman filter *predict* on a generated virtual trajectory to gain the posteriori estimates on $t_2$ (ORU). With the default linear motion analysis, we have the virtual trajectory as

$$\tilde{\mathbf{z}}_t = \mathbf{z}_{t_1} + \frac{t - t_1}{t_2 - t_1}(\mathbf{z}_{t_2} - \mathbf{z}_{t_1}), t_1 < t < t_2. \tag{5.24}$$

Now, to derive the posteriori estimate, we will run the loop between *predict* and *re-update* from $t_1$ to $t_2$.

$$\textit{OC-SORT} \begin{cases} \hat{\mathbf{x}}_{t|t} = \mathbf{F}\hat{\mathbf{x}}_{t-1|t-1} + \mathbf{K}_t(\tilde{\mathbf{z}}_t - \mathbf{H}\mathbf{F}\hat{\mathbf{x}}_{t-1|t-1}) \\ \mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H})(\mathbf{F}\mathbf{P}_{t-1|t-1}\mathbf{F}^\top + \mathbf{Q}_t) \end{cases} \tag{5.25}$$

where the Kalman gain is

$$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^\top(\mathbf{H}\mathbf{P}_{t|t-1}\mathbf{H}^\top + \mathbf{R}_t)^{-1}, \tag{5.26}$$

and we can always rewrite it with

$$\mathbf{P}_{t|t-1} = \mathbf{F}\mathbf{P}_{t-1|t-1}\mathbf{F}^\top + \mathbf{Q}_t. \tag{5.27}$$

In the common practice of Kalman filter, we assume a constant set of Gaussian noise for the process noise $\mathbf{Q}_t$. This assumption typically can't hold in practice. This makes the conflict that when there are consistent observations over time, we require a small process noise for multi-object tracking in high-frame-rate videos. However, when there is a period of observation missing, the direction difference between the true direction and the direction maintained by the linear motion assumption grows. This causes the failure of SORT to consistently track previously lost targets even after re-association.

We show the different outcomes of SORT and OC-SORT upon re-associating lost targets in Equation (5.21) and Equation (5.25). Analyzing their difference more deeply will require more assumptions of the underlying true object trajectory and the observations. Therefore, instead of theoretical proof, we demonstrate the gain of performance from OC-SORT over SORT empirically as shown in the experiments.

# Chapter 6

# Deep OC-SORT: Combine OC-SORT with appearance

As DeepSORT [71] improves SORT [5] by adding the appearance similarity into the association cost calculation, we tried to use appearance similarity as a supplementary term in the association cost calculation to improve the tracking performance. We name this version of implementation Deep OC-SORT [45]. We demonstrate that Deep OC-SORT improves the tracking accuracy on multiple benchmarks. However, these modifications also slow down the running time efficiency of OC-SORT.

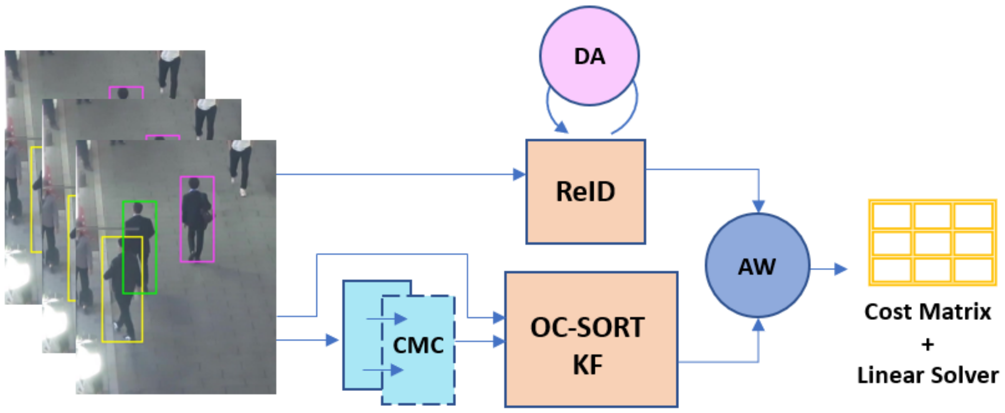


Figure 6.1: Illustration of Deep OC-SORT, incorporating camera motion compensation and appearance similarity to improve OC-SORT.

## 6.1 Method

Now we introduce in details the innovations Deep OC-SORT makes upon OC-SORT. We make three implementation adaptations upon OC-SORT to improve its performance: Camera Motion Compensation (CMC), Dynamic Appearance (DA), and Adaptive Weighting (AW). The pipeline is illustrated in Figure 6.1.

## 6.2 Camera Motion Compensation (CMC)

As OC-SORT is highly dependent on the detection quality, we introduce CMC to more accurately localize objects from frame to frame in moving scenes. Given a scaled rotation matrix $M_t = s_t R_t$ and a translation $T_t$ where $M_t \in \mathbb{R}^{2\times 2}$and $T_t \in \mathbb{R}^{2\times 1}$, we apply them to OC-SORT's three components respectively:

1. **ORU + CMC.** The Kalman filter is updated from the linearly interpolated path, starting at the last known measurement. This last known measurement is comprised of $[x_c, y_c, a, s]$, with the first two entries as the center of the bounding box. The center of the bounding box is similarly transformed by $c \leftarrow M_t c + T_t$, so that the path is interpolated starting from the camera corrected measurement.
2. **OCM + CMC.** Let $p_1, p_2$ be the upper-left and lower-right corner points of a bounding box. OCM uses the last $\Delta t = 3$ bounding boxes to compute a bounding box angular velocity. At each timestep $t$, we apply the transformation $p_i \leftarrow M_t p_i + T_t$ to the bounding box. This goes from $t - \Delta t$ to timestep $t$ during OCM.
3. **OCR + CMC.** For the last-seen bounding box position in OCR, at each timestep $t$, we apply $p_i \leftarrow M_t p_i + T_t$ to adjust its position under CMC.

For OC-SORT, the Kalman state is $\mathbf{x} = [x_c, y_c, a, s, \dot{x_c}, \dot{y_c}, \dot{a}]$. We apply CMC to correct the Kalman state:

$$\begin{cases} \mathbf{x}[0:2] \leftarrow M_t \mathbf{x}[0:2] + T_t, \\ \mathbf{x}[4:6] \leftarrow M_t \mathbf{x}[4:6], \\ P[0:2, 0:2] \leftarrow M_t P[0:2, 0:2] M_t^T, \\ P[4:6, 4:6] \leftarrow M_t P[4:6, 4:6] M_t^T. \end{cases} \tag{6.1}$$

We note that we could apply the *scale* of the CMC transform to the area $a$, or approximate rotation to change the aspect ratio $s$. However, in contrast to the center point, the enclosing bounding box of a rotated object is not approximated linearly and requires a fine-grained mask of the enclosed object. While the approximation works well with OCM and OCR, the Kalman filter is empirically more sensitive to approximate changes. We apply this CMC update before the Kalman extrapolation step so that the *prediction* stage is from the CMC-corrected states.

## 6.3 Dynamic Appearance (DA)

In previous work [1, 20], the deep visual embedding used to describe a tracklet is given by an Exponential Moving Average (EMA) of the deep detection embeddings frame by frame. This requires a weighting factor $\alpha$ to adjust the ratio of the visual embedding from historical and current time steps. We propose to modify the $\alpha$ of the EMA on a per-frame basis, depending on the detector confidence. This flexible $\alpha$ allows selectively incorporating appearance information into a track's model only in high-quality situations.

We use low detector confidence as a proxy to recognize image degradation due to occlusion or blur, allowing us to reject corrupted embeddings. Let $\mathbf{e}_t$ be the tracklet's appearance embedding at time $t$. The standard EMA is

$$\mathbf{e}_t = \alpha \mathbf{e}_{t-1} + (1-\alpha)\mathbf{e}^{\text{new}}, \tag{6.2}$$

where $\mathbf{e}^{\text{new}}$ is the appearance of the matched detection being added to the model. We propose replacing $\alpha$ with a changing $\alpha_t$ defined as

$$\alpha_t = \alpha_f + (1-\alpha_f)(1 - \frac{s_{\text{det}} - \sigma}{1-\sigma}), \tag{6.3}$$

where $s_{\text{det}}$ is the detector confidence, and $\sigma$ is a detection confidence threshold to filter noisy detections, a common practice of previous works [5, 10, 20, 79]. We set the fixed value $\alpha_f = 0.95$. The detector prediction provides $s_{\text{det}}$, controlling the dynamic operation. With $s_{\text{det}} = \sigma$, we have $\alpha_t = 1$, so that the new appearance embedding is totally ignored. In contrast, $s_{\text{det}} = 1$ implies $\alpha_t = \alpha_f$, and $\mathbf{e}^{\text{new}}$ is maximally added to the update of tracklet visual embedding. The value scales linearly with detector confidence. The operation to generate the dynamic appearance introduces no new hyper-parameters to the standard EMA.

## 6.4 Adaptive Weighting (AW)

Our Adaptive Weighting increases the weight of appearance features depending on the discriminativeness of appearance embeddings. Using standard cosine similarity across track and box embeddings results in an $M \times N$ appearance cost matrix, $A_c$ where $M$ and $N$ are the numbers of tracks and detections respectively. $A_c[m, n]$ indicates the entry at the intersection of the $m$-th row and the $n$-th column. This is typically combined with the IoU cost matrix $I_c$ as $C = I_c + a_w A_c$, with a linear sum assignment minimizing cost over $-C$.

We propose to boost individual track-box scores based on discriminativeness, adding $w_b(m,n)$ to the global $a_w$. Let $\tau_m$ represent a track and $d_n$ represent a detection. When $\tau_m$ has a high similarity score to only one box (included in the row $A_c[m,:]$), we increase appearance weight over row $A_c[m,:]$. The same operation is applied to the columns of $A_c$ if a detection $d_n$ is associated discriminatively with only one track. We use $z_{\text{diff}}$ to measure the discriminativeness of box-track pairs, which is defined as the difference between the highest and second-highest values at a row or a column:

$$\begin{aligned} z^{\text{det}}_{\text{diff}}(A_c, n) &= \min(\max_i A_c[i,n] - \max_{j\neq i} A_c[j,n], \epsilon), \\ z^{\text{track}}_{\text{diff}}(A_c, m) &= \min(\max_i A_c[m,i] - \max_{j\neq i} A_c[m,j], \epsilon), \end{aligned} \tag{6.4}$$

where $\epsilon$ is a hyper-parameter to cap the boost where there's a large difference in appearance cost between the first and second best matches. Then, we derive the weighting factor as

$$w_b(m,n) = \left[z^{\text{track}}_{\text{diff}}(A_c, m) + z^{\text{det}}_{\text{diff}}(A_c, n)\right]/2, \tag{6.5}$$

which results in the final cost matrix $C$ as

$$C[m,n] = \text{IoU}[m,n] + [a_w + w_b(m,n)]\, A_c[m,n]. \tag{6.6}$$

We choose to measure the discriminativeness based on only the first and second-highest scores rather than probability distribution metrics like KL divergence, as the spread of values between lower-scoring matches are irrelevant. A true positive appearance match is indicated by one high score having a large distance from the next best match.

## 6.5 Experiments

In this section, we provide experimental evidence to demonstrate the effectiveness of Deep SORT. We also analyze the influence of each module we introduce over OC-SORT [10].

**Datasets and Metrics.** We conduct experiments on multiple datasets to ensure the generalizability of the proposed method. The investigated datasets include the popular pedestrian tracking datasets MOT17 [49], MOT20 [19], and DanceTrack [63]. We follow the HOTA protocol[44] for quantitative evaluation, which provides a more comprehensive measurement of the tracking quality. HOTA is the main metric we refer to for tracking performance. AssA is the metric to measure association accuracy and DetA is for detection accuracy. We also report the metrics in the classic CLEAR protocol [4] for reference where

Table 6.1: Results on MOT17-test and MOT20-test. Methods in the blue blocks share the same detections.

| MOT17 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Tracker | HOTA↑ | MOTA↑ | IDF1↑ | FP($10^4$)↓ | FN($10^4$)↓ | IDs↓ | Frag↓ | AssA↑ | AssR↑ |
| FairMOT [80] | 59.3 | 73.7 | 72.3 | 2.75 | 11.7 | 3,303 | 8,073 | 58.0 | 63.6 |
| TransCt [75] | 54.5 | 73.2 | 62.2 | 2.31 | 12.4 | 4,614 | 9,519 | 49.7 | 54.2 |
| TransTrk [62] | 54.1 | 75.2 | 63.5 | 5.02 | 8.64 | 3,603 | 4,872 | 47.9 | 57.1 |
| GRTU [68] | 62.0 | 74.9 | 75.0 | 3.20 | 10.8 | 1,812 | **1,824** | 62.1 | 65.8 |
| QDTrack [50] | 53.9 | 68.7 | 66.3 | 2.66 | 14.66 | 3,378 | 8,091 | 52.7 | 57.2 |
| MOTR [78] | 57.2 | 71.9 | 68.4 | 2.11 | 13.6 | 2,115 | 3,897 | 55.8 | 59.2 |
| TransMOT [16] | 61.7 | 76.7 | 75.1 | 3.62 | 9.32 | 2,346 | 7,719 | 59.9 | 66.5 |
| ByteTrack [79] | 63.1 | **80.3** | 77.3 | 2.55 | **8.37** | 2,196 | 2,277 | 62.0 | 68.2 |
| OC-SORT [10] | 63.2 | 78.0 | 77.5 | **1.51** | 10.8 | 1,950 | 2,040 | 63.2 | 67.5 |
| StrongSORT [20] | 63.5 | 78.3 | 78.5 | - | - | 1,446 | - | 63.7 | - |
| *StrongSORT++ [20] | 64.4 | 79.6 | 79.5 | 2.79 | 8.62 | 1,194 | **1,866** | 64.4 | **71.0** |
| Deep OC-SORT | **64.9** | 79.4 | **80.6** | 1.66 | 9.88 | **1,023** | 2,196 | **65.9** | 70.1 |
| **MOT20** | | | | | | | | | |
| Tracker | HOTA↑ | MOTA↑ | IDF1↑ | FP($10^4$)↓ | FN($10^4$)↓ | IDs↓ | Frag↓ | AssA↑ | AssR↑ |
| FairMOT [80] | 54.6 | 61.8 | 67.3 | 10.3 | 8.89 | 5,243 | 7,874 | 54.7 | 60.7 |
| Semi-TCL [42] | 55.3 | 65.2 | 70.1 | 6.12 | 11.5 | 4,139 | 8,508 | 56.3 | 60.9 |
| CSTrack [43] | 54.0 | 66.6 | 68.6 | 2.54 | 14.4 | 3,196 | 7,632 | 54.0 | 57.6 |
| GSDT [69] | 53.6 | 67.1 | 67.5 | 3.19 | 13.5 | 3,131 | 9,875 | 52.7 | 58.5 |
| TransMOT [16] | 61.9 | 77.5 | 75.2 | 3.42 | **8.08** | 1,615 | 2,421 | 60.1 | 66.3 |
| ByteTrack [79] | 61.3 | **77.8** | 75.2 | 2.62 | 8.76 | 1,223 | 1,460 | 59.6 | 66.2 |
| OC-SORT [10] | 62.1 | 75.5 | 75.9 | 1.80 | 10.8 | 913 | 1,198 | 62.0 | 67.5 |
| StrongSORT [20] | 61.5 | 72.2 | 75.9 | - | - | 1,066 | - | 63.2 | |
| *StrongSORT++ [20] | 62.6 | 73.8 | 77.0 | **1.66** | 11.8 | **770** | **1,003** | 64.0 | 69.6 |
| Deep OC-SORT | **63.9** | 75.6 | **79.2** | 1.69 | 10.8 | 779 | 1,536 | **65.7** | **70.8** |

* : StrongSORT++ requires offline post-processing while ByteTrack, OC-SORT, StrongSORT and Deep OC-SORT are for online multi-object tracking.

Table 6.2: Results on DanceTrack test set. Methods in the blue block use the same detections.

| Tracker | HOTA↑ | DetA↑ | AssA↑ | MOTA↑ | IDF1↑ |
|---|---|---|---|---|---|
| CenterTrack [83] | 41.8 | 78.1 | 22.6 | 86.8 | 35.7 |
| FairMOT [80] | 39.7 | 66.7 | 23.8 | 82.2 | 40.8 |
| QDTrack [50] | 45.7 | 72.1 | 29.2 | 83.0 | 44.8 |
| TransTrk[62] | 45.5 | 75.9 | 27.5 | 88.4 | 45.2 |
| TraDes [72] | 43.3 | 74.5 | 25.4 | 86.2 | 41.2 |
| MOTR [78] | 54.2 | 73.5 | 40.2 | 79.7 | 51.5 |
| SORT [5] | 47.9 | 72.0 | 31.2 | 91.8 | 50.8 |
| DeepSORT [71] | 45.6 | 71.0 | 29.7 | 87.8 | 47.9 |
| ByteTrack [79] | 47.3 | 71.6 | 31.4 | 89.5 | 52.5 |
| OC-SORT [10] | 55.1 | 80.3 | 38.3 | 92.0 | 54.6 |
| *StrongSORT++ [20] | 55.6 | 80.7 | 38.6 | 91.1 | 55.2 |
| Deep OC-SORT | **61.3** | **82.2** | **45.8** | **92.3** | **61.5** |

* : StrongSORT++ requires offline post-processing while others are for online tracking.

MOTA indicates a overall performance of detection and tracking and IDF1 provides a measurement of association accuracy.

**Implementations** Our implementation is based on OC-SORT [10, 17]. We use the same YOLOX detector as recent works [1, 11, 20, 79] to make a fair comparison of tracking performance. For Re-ID, we use SBS50 from the fast-reid [30] library. For CMC, we adopt the OpenCV contrib VidStab module to generate similarity transforms using feature point extraction, optical flow, and RANSAC, as previous works [1] choose. Across all experiments, we use a fixed $\alpha = 0.95$ for Dynamic Appearance. For experiments on MOT17 and MOT20, we set $a_w = 0.75$ and $\epsilon = 0.5$ for Adaptive Weighting. We use $a_w = 1.25$ for DanceTrack, where we see appearance is more beneficial than IoU, and $\epsilon = 1.0$.

### 6.5.1 Benchmark Results

We conduct experiments on MOT17 [49], MOT20 [19], and DanceTrack [63]. The results on MOT17-test and MOT20-test are shown in Table 6.1. On MOT17-test, Deep OC-SORT achieves 64.9 HOTA, which outperforms all published methods and ranks 2nd on the leaderboard. On MOT20-test, Deep OC-SORT achieves 63.9 HOTA, ranking 1st on the leaderboard. Finally, on the most challenging dataset DanceTrack, where tracking algorithms usually suffer from heavy occlusion and frequent crossovers, our method achieves a new state-of-the-art among published methods as shown in Table 6.2. On all three datasets, using the same detections, our method beats the existing comparisons including SORT [5], DeepSORT [71], ByteTrack [79], OC-SORT [10], and StrongSORT [20]. Being online and without offline post-processing, our method still shows better association accuracy even compared to StrongSORT++, which is the offline version of StrongSORT, enhanced by offline post-processing of tracking trajectories. The benchmark results make strong evidence of the advanced performance of Deep OC-SORT.

Table 6.3: Ablation study on MOT17-val, MOT20-val and DanceTrack-val set.

| | | | | MOT17-val | | | MOT20-val | | | DanceTrack-val | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Appr. | DA | CMC | AW | HOTA↑ | AssA↑ | IDF1↑ | HOTA↑ | AssA↑ | IDF1↑ | HOTA↑ | AssA↑ | IDF1↑ |
| | | | | 68.13 | 70.06 | 79.52 | 58.35 | 56.11 | 74.77 | 53.07 | 35.93 | 52.43 |
| | | ✓ | | 69.66 | 72.72 | 82.44 | 58.33 | 56.12 | 74.63 | 53.57 | 36.68 | 53.30 |
| ✓ | | | | 68.59 | 70.63 | 80.18 | 59.10 | 57.47 | 75.71 | 58.03 | 42.37 | 57.73 |
| ✓ | ✓ | | | 68.65 | 70.85 | 80.45 | 59.16 | 57.60 | 75.87 | 58.36 | 43.00 | 58.17 |
| ✓ | ✓ | ✓ | | 69.80 | 72.86 | 82.56 | 59.35 | 58.00 | 76.11 | 58.46 | 43.33 | 58.83 |
| ✓ | ✓ | ✓ | ✓ | **70.20** | **73.46** | **82.78** | **59.45** | **58.16** | **76.30** | **58.53** | **43.41** | **59.06** |

### 6.5.2 Ablation Study

To demonstrate the effectiveness of the proposed modules, we perform an ablation study on validation sets of MOT17, MOT20, and DanceTrack. With a baseline of OC-SORT, we describe performance with the addition of: Appearance Embedding with a fixed EMA

(Appr.), Dynamic Appearance (DA), Camera Motion Compensation (CMC), and Adaptive Weighting (AW). The results are shown in Table 6.3.

We find that CMC improves performance on MOT17-val and DanceTrack-val sets while providing no improvements on MOT20-val, which is captured from static cameras. Appearance cues (Appr.) improve performance on all datasets across all metrics. Further applying Dynamic Appearance similarly boosts performance on all metrics, while adding no additional hyper-parameters and entirely negligible computation. Finally, Adaptive Weighting provides yet another consistent improvement in performance across all metrics and datasets.

# Chapter 7

# Conclusions

In this paper, we analyze the popular motion-based tracker SORT and recognize its intrinsic limitations from using the Kalman filter. These limitations significantly hurt tracking accuracy when the tracker fails to gain observations for supervision - likely caused by unreliable detectors, occlusion, or fast and non-linear target object motion. To address these issues, we propose *Observation-Centric SORT (OC-SORT)*. OC-SORT is more robust to occlusion and non-linear object motion while keeping simple, online, and real-time. In our experiments on diverse datasets, OC-SORT significantly outperforms the state-of-the-art. The gain is especially significant for multi-object tracking under occlusion and non-linear object motion. OC-SORT is simple, online, and fast, therefore it can provide a basic motion-based multi-object tracking baseline. Also, we demonstrate that OC-SORT is flexible to incorporate other association information, such as appearance similarity, for more accurate multi-object tracking.

# Bibliography

[1] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. Bot-sort: Robust associations multi-pedestrian tracking, 2022. URL `https://arxiv.org/abs/2206.14651`. 6.3, 6.5

[2] Yaakov Bar-Shalom, Fred Daum, and Jim Huang. The probabilistic data association filter. *IEEE Control Systems Magazine*, 29(6):82–100, 2009. 2.2

[3] Sumit Basu, Irfan Essa, and Alex Pentland. Motion regularization for model-based head tracking. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 3, pages 611–616. IEEE, 1996. 4.2.4

[4] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008:1–10, 2008. 6.5

[5] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468. IEEE, 2016. 1, 2.1, 2.2, 3.1, 3.1.1, 3.2.2, 4.3, 6, 6.3, 6.2, 6.5.1

[6] Guillem Brasó and Laura Leal-Taixé. Learning a neural solver for multiple object tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6247–6257, 2020. 4.5, 4.8

[7] Jiarui Cai, Mingze Xu, Wei Li, Yuanjun Xiong, Wei Xia, Zhuowen Tu, and Stefano Soatto. Memot: multi-object tracking with memory. In *CVPR*, 2022. 4.1, 4.2

[8] Jinkun Cao, Jiangmiao Pang, Xinshuo Weng, Rawal Khirodkar, and Kris M Kitani. Object tracking by hierarchical part-whole attention. 4.2.4

[9] Jinkun Cao, Hongyang Tang, Hao-Shu Fang, Xiaoyong Shen, Cewu Lu, and Yu-Wing Tai. Cross-domain adaptation for animal pose estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9498–9507, 2019. 4.2.4

[10] Jinkun Cao, Xinshuo Weng, Rawal Khirodkar, Jiangmiao Pang, and Kris Kitani. Observation-centric sort: Rethinking sort for robust multi-object tracking. *arXiv preprint arXiv:2203.14360*, 2022. 6.3, 6.5, 6.1, 6.2, 6.5, 6.5.1

[11] Jinkun Cao, Hao Wu, and Kris Kitani. Track targets by dense spatio-temporal position encoding. *arXiv preprint arXiv:2210.09455*, 2022. 6.5

[12] Jinkun Cao, Hao Wu, and Kris Kitani. Track targets by dense spatio-temporal position encoding. *arXiv preprint arXiv:2210.09455*, 2022. 2.2, 4.1, 4.3

[13] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse:

3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019. 4.2.4

[14] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11(4), 2010. 4.3

[15] Wongun Choi. Near-online multi-target tracking with aggregated local flow descriptor. In *Proceedings of the IEEE international conference on computer vision*, pages 3029–3037, 2015. 2.1, 4.5

[16] Peng Chu, Jiang Wang, Quanzeng You, Haibin Ling, and Zicheng Liu. Transmot: Spatial-temporal graph transformer for multiple object tracking. *arXiv preprint arXiv:2104.00194*, 2021. 4.1, 4.2, 6.1

[17] MMTracking Contributors. MMTracking: OpenMMLab video perception toolbox and benchmark. https://github.com/open-mmlab/mmtracking, 2020. 6.5

[18] Peng Dai, Renliang Weng, Wongun Choi, Changshui Zhang, Zhangping He, and Wei Ding. Learning a proposal classifier for multiple object tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2443–2452, 2021. 4.8

[19] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, and Laura Leal-Taixé. Mot20: A benchmark for multi object tracking in crowded scenes. *arXiv preprint arXiv:2003.09003*, 2020. (document), 4.1, 5.2, 6.5, 6.5.1

[20] Yunhao Du, Zhicheng Zhao, Yang Song, Yanyun Zhao, Fei Su, Tao Gong, and Hongying Meng. Strongsort: Make deepsort great again. *IEEE Transactions on Multimedia*, pages 1–14, 2023. doi: 10.1109/TMM.2023.3240881. 6.3, 6.3, 6.1, 6.2, 6.5, 6.5.1

[21] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014. 5.2.2, 5.2.2

[22] Hao-Shu Fang, Jinkun Cao, Yu-Wing Tai, and Cewu Lu. Pairwise body-part attention for recognizing human-object interactions. In *Proceedings of the European conference on computer vision (ECCV)*, pages 51–67, 2018. 4.2.4

[23] Juergen Gall, Angela Yao, Nima Razavi, Luc Van Gool, and Victor Lempitsky. Hough forests for object detection, tracking, and action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 33(11):2188–2202, 2011. 4.2.4

[24] Damien Garreau, Wittawat Jitkrittum, and Motonobu Kanagawa. Large sample analysis of the median heuristic. *arXiv preprint arXiv:1707.07269*, 2017. 5.2.3

[25] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021. 4.1, 4.2.2

[26] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11): 1231–1237, 2013. 4.1

[27] Nicolas Franco Gonzalez, Andres Ospina, and Philippe Calvez. Smat: Smart multiple affinity metrics for multiple object tracking. In *International Conference on Image*

*Analysis and Recognition*, pages 48–62. Springer, 2020. 4.5

[28] Fredrik Gustafsson, Fredrik Gunnarsson, Niclas Bergman, Urban Forssell, Jonas Jansson, Rickard Karlsson, and P-J Nordlund. Particle filters for positioning, navigation, and tracking. *IEEE Transactions on signal processing*, 50(2):425–437, 2002. 2.1

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2.2

[30] Lingxiao He, Xingyu Liao, Wu Liu, Xinchen Liu, Peng Cheng, and Tao Mei. Fastreid: A pytorch toolbox for general instance re-identification. *arXiv preprint arXiv:2006.02631*, 2020. 6.5

[31] David V Hinkley. On the ratio of two correlated normal random variables. *Biometrika*, 56(3):635–639, 1969. 5.1

[32] Andrea Hornakova, Roberto Henschel, Bodo Rosenhahn, and Paul Swoboda. Lifted disjoint paths with application in multiple object tracking. In *International Conference on Machine Learning*, pages 4364–4375. PMLR, 2020. 4.7

[33] Andrea Hornakova, Timo Kaiser, Paul Swoboda, Michal Rolinek, Bodo Rosenhahn, and Roberto Henschel. Making higher order mot scalable: An efficient approximate solver for lifted disjoint paths. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6330–6340, 2021. 4.8

[34] Andrew H Jazwinski. *Stochastic processes and filtering theory.* Courier Corporation, 2007. 4.3

[35] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997. 2.1, 5.2.1

[36] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. 5.2.1

[37] Rudolf Emil Kalman et al. Contributions to the theory of optimal control. *Bol. soc. mat. mexicana*, 5(2):102–119, 1960. 2.1, 3.1.1, 5.2.1

[38] Kris M Kitani, Brian D Ziebart, James Andrew Bagnell, and Martial Hebert. Activity forecasting. In *European conference on computer vision*, pages 201–214. Springer, 2012. 4.2.4

[39] Jonathan Ko and Dieter Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009. 5.2.1

[40] Parth Kothari, Sven Kreiss, and Alexandre Alahi. Human trajectory forecasting in crowds: A deep learning perspective. *IEEE Transactions on Intelligent Transportation Systems*, 2021. 4.2.4

[41] Erich L Lehmann and George Casella. *Theory of point estimation.* Springer Science & Business Media, 2006. 2.1

[42] Wei Li, Yuanjun Xiong, Shuo Yang, Mingze Xu, Yongxin Wang, and Wei Xia. Semi-tcl: Semi-supervised track contrastive representation learning. *arXiv preprint*

*arXiv:2107.02396*, 2021. 4.2, 6.1

[43] Chao Liang, Zhipeng Zhang, Yi Lu, Xue Zhou, Bing Li, Xiyong Ye, and Jianxiao Zou. Rethinking the competition between detection and reid in multi-object tracking. *arXiv preprint arXiv:2010.12138*, 2020. 4.2, 6.1

[44] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. Hota: A higher order metric for evaluating multi-object tracking. *International journal of computer vision*, 129(2):548–578, 2021. 4.1, 6.5

[45] Gerard Maggiolino, Adnan Ahmad, Jinkun Cao, and Kris Kitani. Deep oc-sort: Multi-pedestrian tracking by adaptive re-identification. *arXiv preprint arXiv:2302.11813*, 2023. 6

[46] Gerard Maggiolino, Adnan Ahmad, Jinkun Cao, and Kris Kitani. Deep oc-sort: Multi-pedestrian tracking by adaptive re-identification. *arXiv preprint arXiv:2302.11813*, 2023. 4.2.3

[47] Tim Meinhardt, Alexander Kirillov, Laura Leal-Taixe, and Christoph Feichtenhofer. Trackformer: Multi-object tracking with transformers. *arXiv preprint arXiv:2101.02702*, 2021. 2.2, 4.7

[48] Stan Melax, Leonid Keselman, and Sterling Orsten. Dynamics based 3d skeletal hand tracking. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 184–184, 2013. 4.2.4

[49] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016. 3.1.2, 4.1, 6.5, 6.5.1

[50] Jiangmiao Pang, Linlu Qiu, Xia Li, Haofeng Chen, Qi Li, Trevor Darrell, and Fisher Yu. Quasi-dense similarity learning for multiple object tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 164–173, 2021. 2.2, 4.1, 4.3, 4.7, 6.1, 6.2

[51] Dezhi Peng, Zikai Sun, Zirong Chen, Zirui Cai, Lele Xie, and Lianwen Jin. Detecting heads using feature refine net and cascaded multi-scale architecture. *arXiv preprint arXiv:1803.09256*, 2018. 4.2.4

[52] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986. 2.1

[53] Akshay Rangesh, Pranav Maheshwari, Mez Gebre, Siddhesh Mhatre, Vahid Ramezani, and Mohan M Trivedi. Trackmpnn: A message passing graph neural architecture for multi-object tracking. *arXiv preprint arXiv:2101.04206*, 2021. 4.5

[54] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2.2

[55] Steven Reece and Stephen Roberts. An introduction to gaussian processes for the kalman filter expert. In *2010 13th International Conference on Information Fusion*, pages 1–9. IEEE, 2010. 5.2.1

[56] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-

time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 2.2, 3.1.2

[57] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 658–666, 2019. 4.2.3

[58] Toby Sharp, Cem Keskin, Duncan Robertson, Jonathan Taylor, Jamie Shotton, David Kim, Christoph Rhemann, Ido Leichter, Alon Vinnikov, Yichen Wei, et al. Accurate, robust, and flexible real-time hand tracking. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 3633–3642, 2015. 4.2.4

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2.2, 3.1.2

[60] Gerald L Smith, Stanley F Schmidt, and Leonard A McGee. *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*. National Aeronautics and Space Administration, 1962. 2.1, 5.2.1

[61] Daniel Stadler and Jurgen Beyerer. Improving multiple pedestrian tracking by track management and occlusion handling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10958–10967, 2021. 4.8

[62] Peize Sun, Jinkun Cao, Yi Jiang, Rufeng Zhang, Enze Xie, Zehuan Yuan, Changhu Wang, and Ping Luo. Transtrack: Multiple object tracking with transformer. *arXiv preprint arXiv:2012.15460*, 2020. 2.2, 4.1, 4.3, 6.1, 6.2

[63] Peize Sun, Jinkun Cao, Yi Jiang, Zehuan Yuan, Song Bai, Kris Kitani, and Ping Luo. Dancetrack: Multi-object tracking in uniform appearance and diverse motion. *arXiv preprint arXiv:2111.14690*, 2021. (document), 1.1, 4.1, 4.2.2, 6.5, 6.5.1

[64] Ramana Sundararaman, Cedric De Almeida Braga, Eric Marchand, and Julien Pettre. Tracking pedestrian heads in dense crowd. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3865–3875, 2021. (document), 4.1, 4.6, 4.2.4, 5.2

[65] Pavel Tokmakov, Jie Li, Wolfram Burgard, and Adrien Gaidon. Learning to track with object permanence. In *ICCV*, pages 10860–10869, 2021. (document), 4.1, 4.1, 4.5, 4.2.3

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2.2

[67] Gaoang Wang, Renshu Gu, Zuozhu Liu, Weijie Hu, Mingli Song, and Jenq-Neng Hwang. Track without appearance: Learn box and tracklet embedding with local and global motion patterns for vehicle tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9876–9886, 2021. 4.5

[68] Shuai Wang, Hao Sheng, Yang Zhang, Yubin Wu, and Zhang Xiong. A general recurrent tracking framework without real data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13219–13228, 2021. 4.1, 6.1

[69] Yongxin Wang, Kris Kitani, and Xinshuo Weng. Joint object detection and multi-object

tracking with graph neural networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13708–13715. IEEE, 2021. 4.2, 6.1

[70] Christopher Williams and Carl Rasmussen. Gaussian processes for regression. *Advances in neural information processing systems*, 8, 1995. 5.2.1

[71] 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. 2.1, 2.2, 4.3, 4.2.3, 6, 6.2, 6.5.1

[72] Jialian Wu, Jiale Cao, Liangchen Song, Yu Wang, Ming Yang, and Junsong Yuan. Track to detect and segment: An online multi-object tracker. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12352–12361, 2021. 4.3, 6.2

[73] Yu Xiang, Alexandre Alahi, and Silvio Savarese. Learning to track: Online multi-object tracking by decision making. In *Proceedings of the IEEE international conference on computer vision*, pages 4705–4713, 2015. 4.5

[74] Yuliang Xiu, Jiefeng Li, Haoyu Wang, Yinghong Fang, and Cewu Lu. Pose flow: Efficient online pose tracking. *arXiv preprint arXiv:1802.00977*, 2018. 4.2.4

[75] Yihong Xu, Yutong Ban, Guillaume Delorme, Chuang Gan, Daniela Rus, and Xavier Alameda-Pineda. Transcenter: Transformers with dense queries for multiple-object tracking. *arXiv preprint arXiv:2103.15145*, 2021. 4.1, 4.2, 4.7, 4.8, 6.1

[76] Bin Yan, Yi Jiang, Peize Sun, Dong Wang, Zehuan Yuan, Ping Luo, and Huchuan Lu. Towards grand unification of object tracking. In *ECCV*. Springer, 2022. 4.1

[77] Fangao Zeng and et al. Motr: End-to-end multiple-object tracking with transformer. In *ECCV*. Springer, 2022. 2.2, 4.1, 4.3

[78] Fangao Zeng, Bin Dong, Tiancai Wang, Xiangyu Zhang, and Yichen Wei. Motr: End-to-end multiple-object tracking with transformer. *arXiv preprint arXiv:2105.03247*, 2021. 6.1, 6.2

[79] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box. *arXiv preprint arXiv:2110.06864*, 2021. 2.1, 4.1, 4.1, 4.2, 4.3, 4.2.1, 4.2.4, 6.3, 6.1, 6.2, 6.5, 6.5.1

[80] 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, 2021. (document), 2.2, 4.1, 4.2, 4.3, 4.6, 4.2.4, 6.1, 6.2

[81] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020. 4.2.3

[82] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 2.2

[83] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points. In *ECCV*, pages 474–490. Springer, 2020. 2.1, 4.3, 4.5, 4.7, 4.2.3, 4.3, 6.2

[84] Xingyi Zhou, Tianwei Yin, Vladlen Koltun, and Philipp Krähenbühl. Global tracking transformers. In *CVPR*, 2022. 4.1, 4.3