Introduction to Object Tracking Algorithms

Mehmet Kerem Turkcan

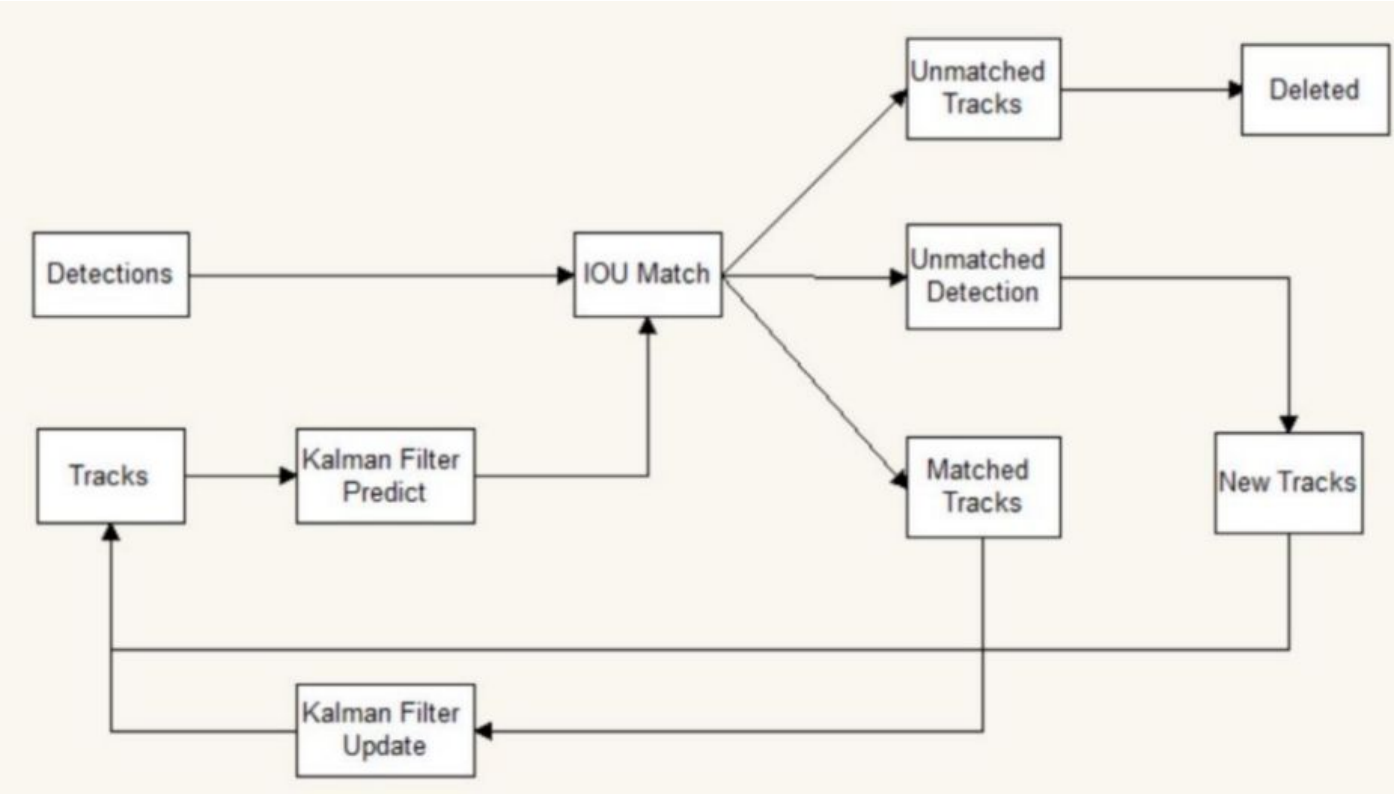
Multi-Object Tracking

Multi-Object Tracking

- **Multi-object tracking (MOT)** aims to detect and estimate the spatial-temporal trajectories of multiple objects in a video stream.
- Currently, **tracking-by-detection** has become the most effective paradigm for the MOT task.
  - Tracking-by-detection contains an object detection step, followed by a tracking step. The tracking step is usually built from two main parts:
    i. Motion model and state estimation for predicting the bounding boxes of the tracklets in the following frames. A **Kalman filter (KF)** is the popular choice for this task.
    ii. Associating the new frame detections with the current set of tracks. Two leading approaches are used for tackling the association task:
      1. Localization of the object, mainly intersection-over-union (IoU) between the predicted tracklet bounding box and the detection bounding box.
      2. Appearance model of the object and solving a re-identification (Re-ID) task.

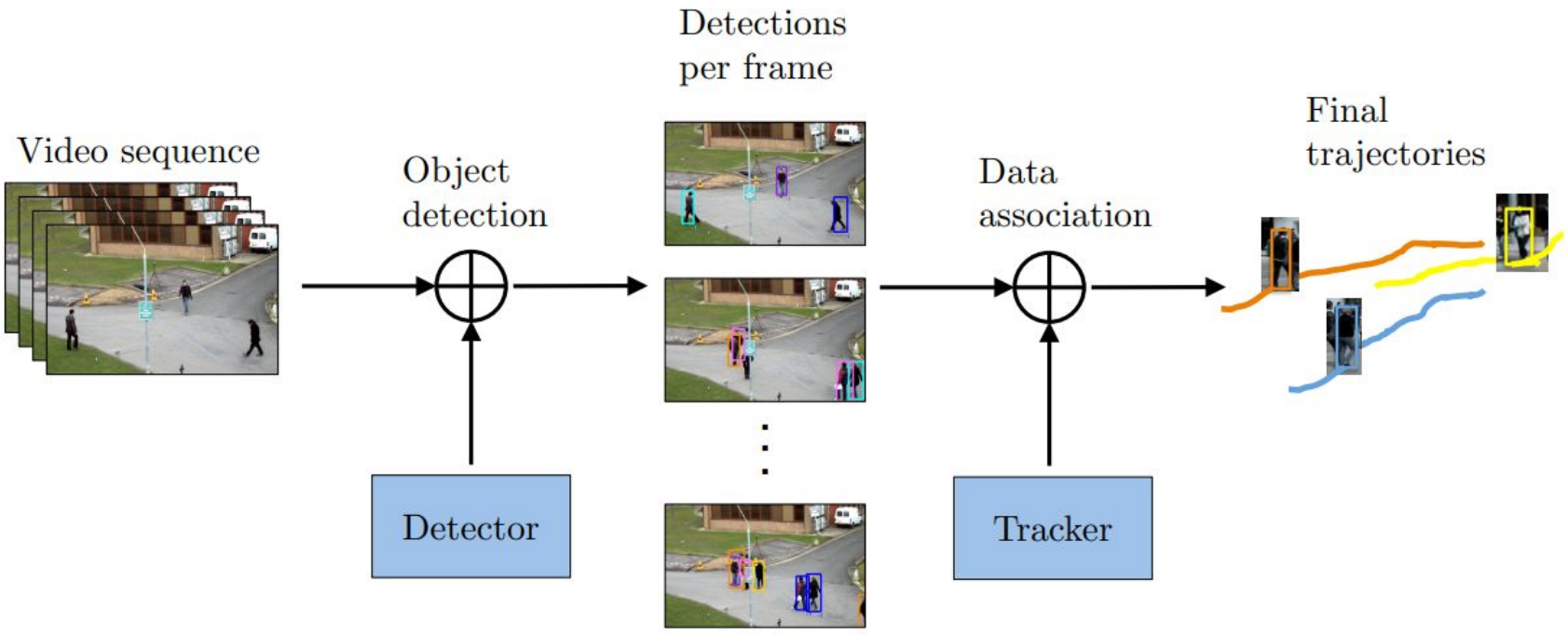
    Both approaches are quantified into distances.

Multi-Object Tracking: Tracks



https://www.mdpi.com/2076-3417/11/12/5619

Tracking-by-Detection Paradigm



https://arxiv.org/abs/1411.7935

Kalman Filter

- **Let us consider** a bounding box under tracking as being described by 7 quantities:

  [x,y,s,r,x',y',s']

  where s is the area of the bounding box, (x,y) its center, r is its aspect ratio.

- We assume the aspect ratio of an object does not change linearly during its observation in a video, but the other quantities possibly do.
- The Kalman filter infers our first order values (x',y',s'). We make two assumptions:
  - Our sensors are noisy and their output and noise can be modeled by Gaussians.
  - Our initial prior on the state is also a Gaussian.
- Given the Kalman filter "update", in the next step, we can "predict" x <= x + x' and so on (dt=1).
- However, our new measurements will necessitate updates of our estimates of x', y' and s'.

Kalman Filter

The state of the filter is represented by two variables:

- $\mathbf{x}_{k|k}$, the *a posteriori* state estimate at time *k* given observations up to and including at time *k*;
- $\mathbf{P}_{k|k}$, the *a posteriori* estimate covariance matrix (a measure of the estimated accuracy of the state estimate).

In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, one must model the process in accordance with the following framework. This means specifying the matrices, for each time-step *k*, following:

- $\mathbf{F}_k$, the state-transition model;
- $\mathbf{H}_k$, the observation model;
- $\mathbf{Q}_k$, the covariance of the process noise;
- $\mathbf{R}_k$, the covariance of the observation noise;
- and sometimes $\mathbf{B}_k$, the control-input model as described below; if $\mathbf{B}_k$ is included, then there is also
- $\mathbf{u}_k$, the control vector, representing the controlling input into control-input model.

Predict [edit]

Predicted (*a priori*) state estimate	$\hat{\mathbf{x}}_{k\|k-1} = \mathbf{F}_k\mathbf{x}_{k-1\|k-1} + \mathbf{B}_k\mathbf{u}_k$
Predicted (*a priori*) estimate covariance	$\hat{\mathbf{P}}_{k\|k-1} = \mathbf{F}_k\mathbf{P}_{k-1\|k-1}\mathbf{F}_k^{\mathsf{T}} + \mathbf{Q}_k$

Update [edit]

Innovation or measurement pre-fit residual	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\|k-1}$
Innovation (or pre-fit residual) covariance	$\mathbf{S}_k = \mathbf{H}_k\hat{\mathbf{P}}_{k\|k-1}\mathbf{H}_k^{\mathsf{T}} + \mathbf{R}_k$
Optimal Kalman gain	$\mathbf{K}_k = \hat{\mathbf{P}}_{k\|k-1}\mathbf{H}_k^{\mathsf{T}}\mathbf{S}_k^{-1}$
Updated (*a posteriori*) state estimate	$\mathbf{x}_{k\|k} = \hat{\mathbf{x}}_{k\|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$
Updated (*a posteriori*) estimate covariance	$\mathbf{P}_{k\|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\,\hat{\mathbf{P}}_{k\|k-1}$
Measurement post-fit residual	$\tilde{\mathbf{y}}_{k\|k} = \mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k\|k}$

https://en.wikipedia.org/wiki/Kalman_filter

Kalman Filter

The Kalman filter model assumes the true state at time k is evolved from the state at $(k-1)$ according to

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k$$

where

- $\mathbf{F}_k$ is the state transition model which is applied to the previous state $\mathbf{x}_{k-1}$;
- $\mathbf{B}_k$ is the control-input model which is applied to the control vector $\mathbf{u}_k$;
- $\mathbf{w}_k$ is the process noise, which is assumed to be drawn from a zero mean multivariate normal distribution, $\mathcal{N}$, with covariance, $\mathbf{Q}_k$: $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)$.

At time k an observation (or measurement) $\mathbf{z}_k$ of the true state $\mathbf{x}_k$ is made according to

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k$$

where

- $\mathbf{H}_k$ is the observation model, which maps the true state space into the observed space and
- $\mathbf{v}_k$ is the observation noise, which is assumed to be zero mean Gaussian white noise with covariance $\mathbf{R}_k$: $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)$.

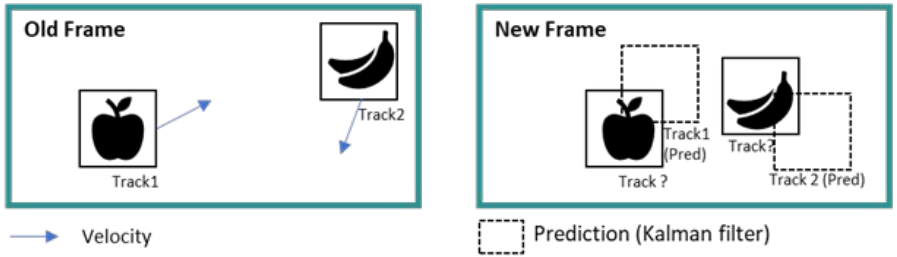
Simple Online and Realtime Tracking (SORT)

Stage 1: Detection

How to identify individual objects/tracks?

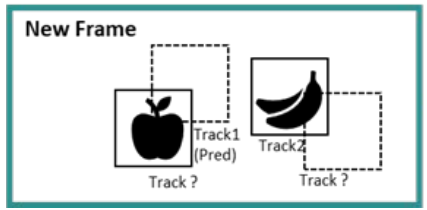
Stage 2: Kalman Filter

How to predict where track will move?



Stage 3: Hungarian Algorithm

What track do the detected objects belong to?



2.1. Calculate IOU Matrix

	Apple (new frame)	Banana (new frame)
Track1 (Pred)	0.2	0
Track2 (Pred)	0	0.15

2.2. Hungarian Algorithm

Goal: Algorithm to solve the „assignment problem". It will asign tracks to detection while optimizing for the global cost.

Constraint: Minimum treshold

2.3. Asignment

	Apple (new frame)	Banana (new frame)
Track1 (Pred)	0.2	0
Track2 (Pred)	0	0.15

https://www.linkedin.com/pulse/object-tracking-sort-deepsort-daniel-pleus/

Simple Online and Realtime Tracking (SORT)

SORT presents a simple approach to tracking that's straightforward to implement:

1. For each detected object, we create a Kalman filter-based tracker. We use this tracker to predict where the object will be in the next frame.
2. We run our object detection algorithm on the new frame.
3. We use the Hungarian algorithm to match objects to trackers, and update trackers with that information.
4. We create new trackers for new objects.
5. We destroy trackers whose objects haven't been seen recently ("stale").

SORT: Hungarian Algorithm for Assignment

```
def linear_assignment(cost_matrix):
  try:
    import lap
    _, x, y = lap.lapjv(cost_matrix, extend_cost=True)
    return np.array([[y[i],i] for i in x if i >= 0]) #
  except ImportError:
    from scipy.optimize import linear_sum_assignment
    x, y = linear_sum_assignment(cost_matrix)
    return np.array(list(zip(x, y)))
```

SORT: State Representation

```
def convert_bbox_to_z(bbox):
  """
  Takes a bounding box in the form [x1,y1,x2,y2] and returns z in the form
    [x,y,s,r] where x,y is the centre of the box and s is the scale/area and r is
    the aspect ratio
  """
  w = bbox[2] - bbox[0]
  h = bbox[3] - bbox[1]
  x = bbox[0] + w/2.
  y = bbox[1] + h/2.
  s = w * h    #scale is just area
  r = w / float(h)
  return np.array([x, y, s, r]).reshape((4, 1))
```

SORT: Kalman Filter Setup

```
class KalmanBoxTracker(object):
  """
  This class represents the internal state of individual tracked objects observed as bbox.
  """
  count = 0
  def __init__(self,bbox):
    """
    Initialises a tracker using initial bounding box.
    """
    #define constant velocity model
    self.kf = KalmanFilter(dim_x=7, dim_z=4)
    self.kf.F = np.array([[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]])
    self.kf.H = np.array([[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]])

    self.kf.R[2:,2:] *= 10.
    self.kf.P[4:,4:] *= 1000. #give high uncertainty to the unobservable initial velocities
    self.kf.P *= 10.
    self.kf.Q[-1,-1] *= 0.01
    self.kf.Q[4:,4:] *= 0.01

    self.kf.x[:4] = convert_bbox_to_z(bbox)
```

- $\mathbf{F}_k$, the state-transition model;
- $\mathbf{H}_k$, the observation model;
- $\mathbf{Q}_k$, the covariance of the process noise;
- $\mathbf{R}_k$, the covariance of the observation noise;

SORT: Kalman Filter Update

```
def update(self,bbox):
  """
  Updates the state vector with observed bbox.
  """
  self.time_since_update = 0
  self.history = []
  self.hits += 1
  self.hit_streak += 1
  self.kf.update(convert_bbox_to_z(bbox))
```

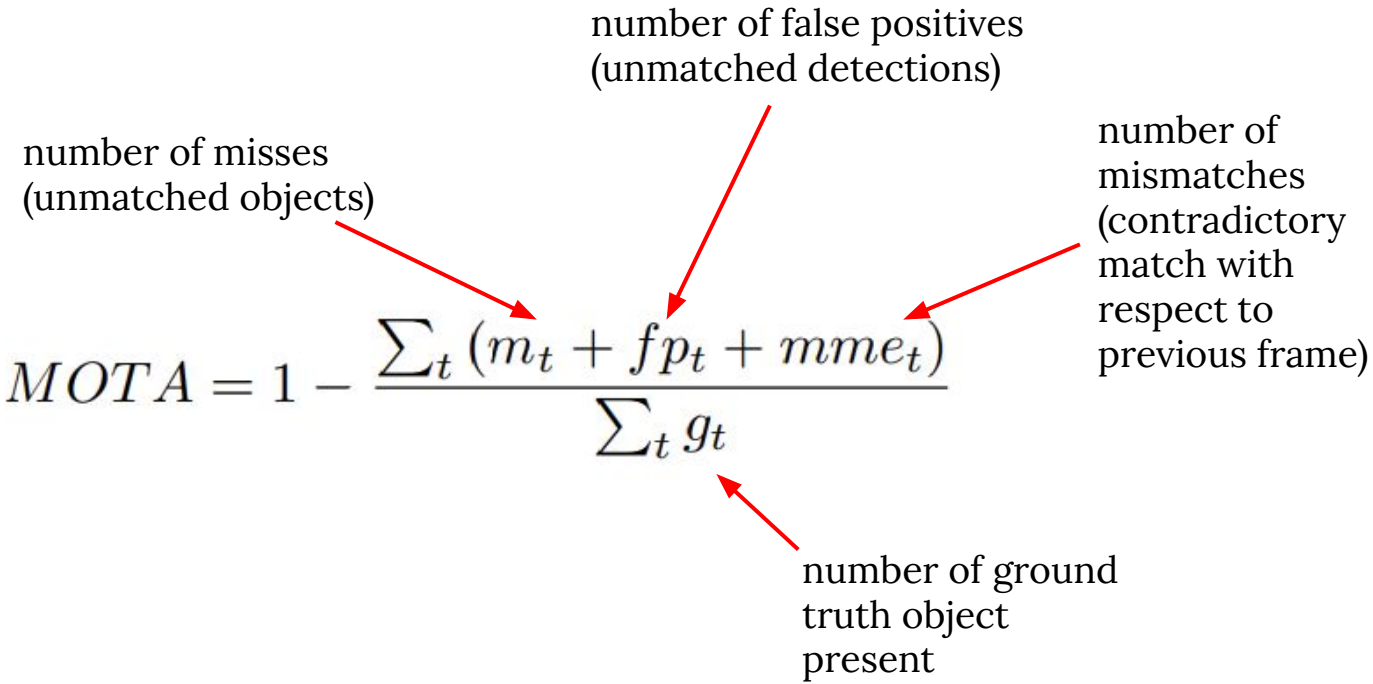
SORT: Kalman Filter Predict

```
def predict(self):
  """
  Advances the state vector and returns the predicted bounding box estimate.
  """
  if((self.kf.x[6]+self.kf.x[2])<=0):
    self.kf.x[6] *= 0.0
  self.kf.predict()
  self.age += 1
  if(self.time_since_update>0):
    self.hit_streak = 0
  self.time_since_update += 1
  self.history.append(convert_x_to_bbox(self.kf.x))
  return self.history[-1]
```

SORT: Kalman Filter Predict

```
def predict(self):
  """
  Advances the state vector and returns the predicted bounding box estimate.
  """
  if((self.kf.x[6]+self.kf.x[2])<=0):
    self.kf.x[6] *= 0.0
  self.kf.predict()
  self.age += 1
  if(self.time_since_update>0):
    self.hit_streak = 0
  self.time_since_update += 1
  self.history.append(convert_x_to_bbox(self.kf.x))
  return self.history[-1]
```

Evaluation: Multiple Object Tracking Accuracy



Evaluation: Multiple Object Tracking Precision

$$MOTP = \frac{\sum_{i,t} d_{i,t}}{\sum_t c_t}$$

position error for a matched object-hypothesis pair (usually, IoU)

number of matches made

Evaluation: IDF1

IDF1 calculates a **bijective** (one-to-one) **mapping** between the sets of gtTrajs and prTrajs.

Its definition requires some new terms:

- **IDTPs** (identity true positives) are matches on the overlapping part (with valid detection thresholds) of trajectories that are matched together.
- **IDFNs** (identity false negatives) and **IDFPs** (identity false positives) are the remaining gtDets and prDets respectively, from both non-overlapping sections of matched trajectories, and from the remaining trajectories that are not matched.

The matching is performed such that IDF1 is optimised. This is performed by enumerating the number of IDFPs and FDFNs that would result from each match (non-overlapping sections), and from nonmatched trajectories (number of prDet and gtDet in these trajectories, respectively). The Hungarian algorithm is used to select which trajectories to match so that the sum of the number of IDFPs and IDFNs is minimised.

Evaluation: IDF1 (Identification F1 Score)

$$\text{ID-Recall} = \frac{|\text{IDTP}|}{|\text{IDTP}| + |\text{IDFN}|}$$

$$\text{ID-Precision} = \frac{|\text{IDTP}|}{|\text{IDTP}| + |\text{IDFP}|}$$

$$\text{IDF1} = \frac{|\text{IDTP}|}{|\text{IDTP}| + 0.5\,|\text{IDFN}| + 0.5\,|\text{IDFP}|}$$

Evaluation: IDF1

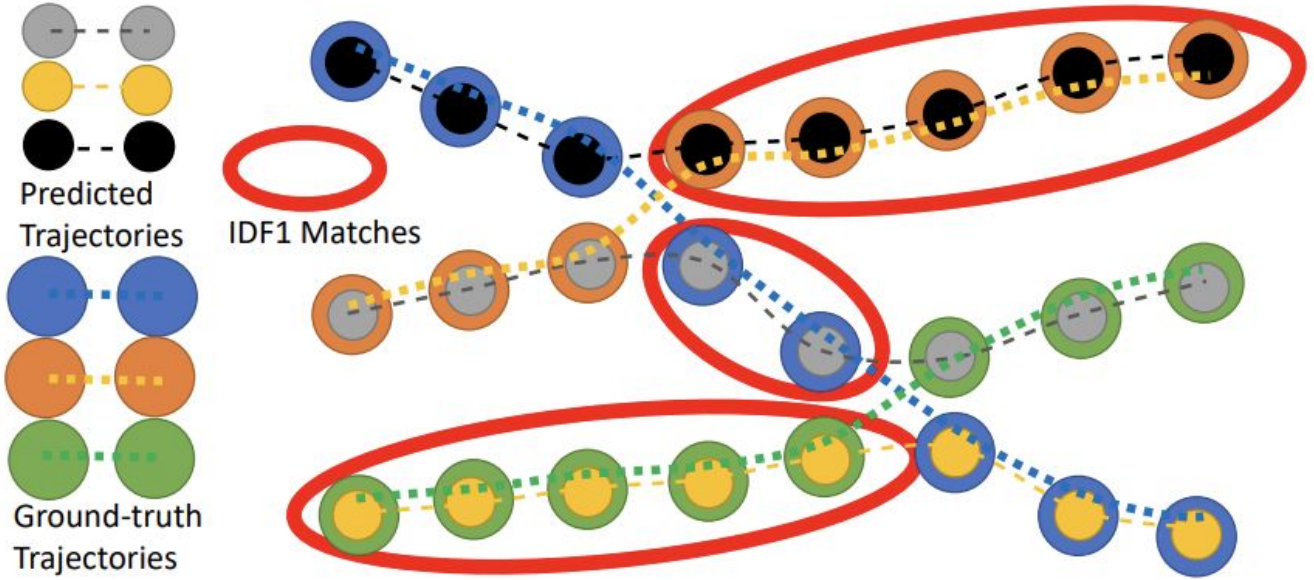
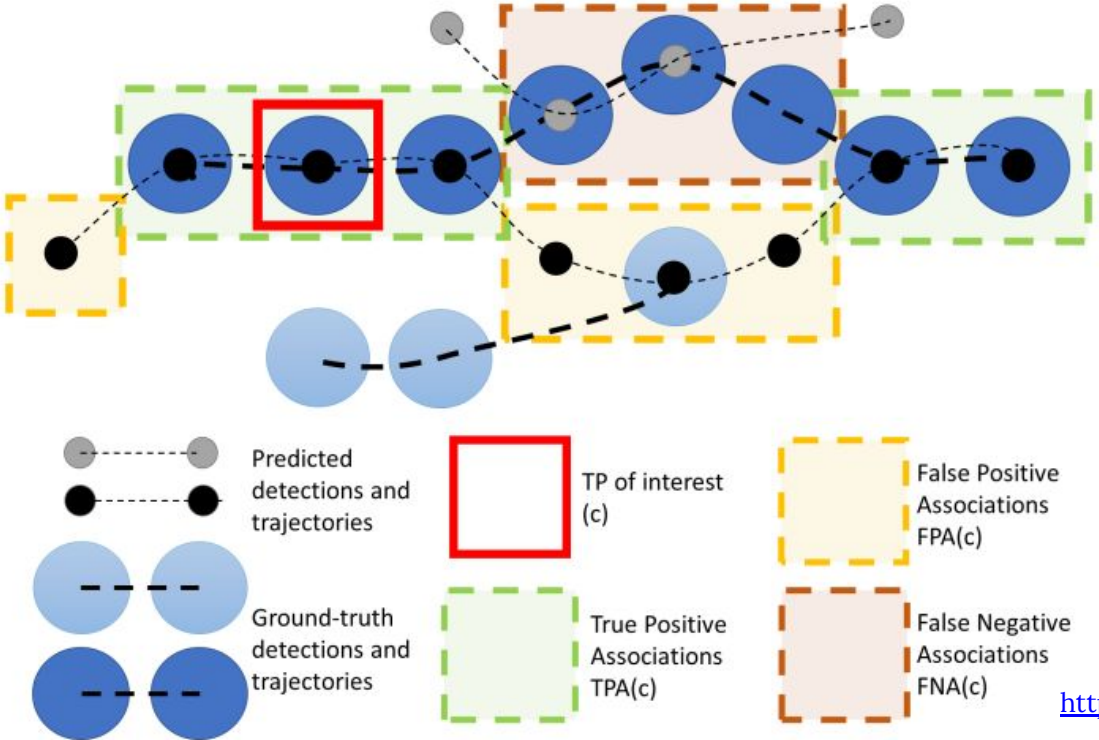


Fig. 10 A tracking example which shows how the single best trajectory matching, as performed by IDF1, can result in unintuitive matches between trajectories.

Evaluation: Higher Order Tracking Accuracy (HOTA)

HOTA calculates a **bijective** (one-to-one) **mapping** between the sets of gtDets and prDets per frame. Given the matching, for each matched pair corresponding to a TP with a given IoU threshold, we calculate the following quantities:



https://arxiv.org/abs/2009.07736

Evaluation: Higher Order Tracking Accuracy (HOTA)

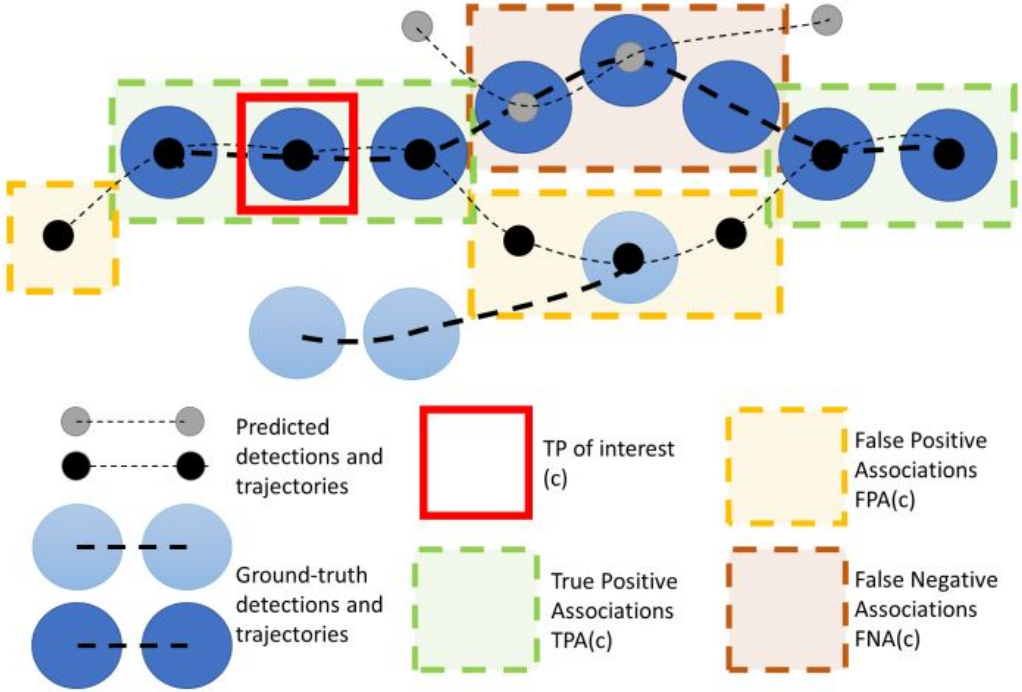


Fig. 2 A visual explanation of the concepts of TPA, FPA and FNA. The different TPAs, FPAs and FNAs are highlighted for the TP of interest c. The TPAs (green) for c (red) are the matches which have the same prID and the same gtID. The FPAs have the same prID but either a different or no gtID. The FNAs have the same gtID but either a different or no prID. In the diagram, *c* has five TPAs, four FPAs and three FNAs. Conceptually these concepts are trying to answer the question: *For the matched TP c, how accurate is the alignment between the gtTraj for this TP (large dark blue circles) and the prTraj for this TP (small black circles).*

https://github.com/JonathonLuiten/TrackEval/blob/master/trackeval/metrics/hota.py

Evaluation: Higher Order Tracking Accuracy (HOTA)

$$\mathrm{TPA}(c) = \{k\},$$
$$k \in \{\mathrm{TP} \mid \mathrm{prID}(k) = \mathrm{prID}(c) \wedge \mathrm{gtID}(k) = \mathrm{gtID}(c)\}$$

$$\mathrm{FNA}(c) = \{k\},$$
$$k \in \begin{matrix}\{\mathrm{TP} \mid \mathrm{prID}(k) \neq \mathrm{prID}(c) \wedge \mathrm{gtID}(k) = \mathrm{gtID}(c)\} \\ \cup \{\mathrm{FN} \mid \mathrm{gtID}(k) = \mathrm{gtID}(c)\}\end{matrix}$$

$$\mathrm{FPA}(c) = \{k\},$$
$$k \in \begin{matrix}\{\mathrm{TP} \mid \mathrm{prID}(k) = \mathrm{prID}(c) \wedge \mathrm{gtID}(k) \neq \mathrm{gtID}(c)\} \\ \cup \{\mathrm{FP} \mid \mathrm{prID}(k) = \mathrm{prID}(c)\}.\end{matrix}$$

$$\mathrm{HOTA}_\alpha = \sqrt{\frac{\sum_{c \in \{\mathrm{TP}\}} \mathcal{A}(c)}{|\mathrm{TP}| + |\mathrm{FN}| + |\mathrm{FP}|}}$$

$$\mathcal{A}(c) = \frac{|\mathrm{TPA}(c)|}{|\mathrm{TPA}(c)| + |\mathrm{FNA}(c)| + |\mathrm{FPA}(c)|}$$

$$\mathrm{HOTA} = \int_0^1 \mathrm{HOTA}_\alpha \, d\alpha \approx \frac{1}{19} \sum_{\alpha \in \{\substack{0.05,\ 0.1,\ \ldots \\ 0.9,\ 0.95}\}} \mathrm{HOTA}_\alpha$$