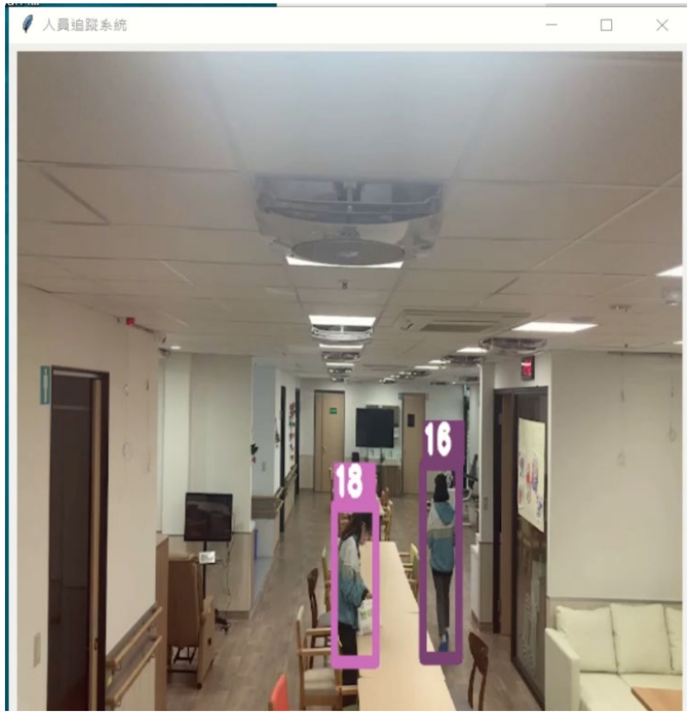
Computer vision tasks and corresponding NNs

SlowFast

SORT, ByteTrack
DeepSORT, JDE

Action classification

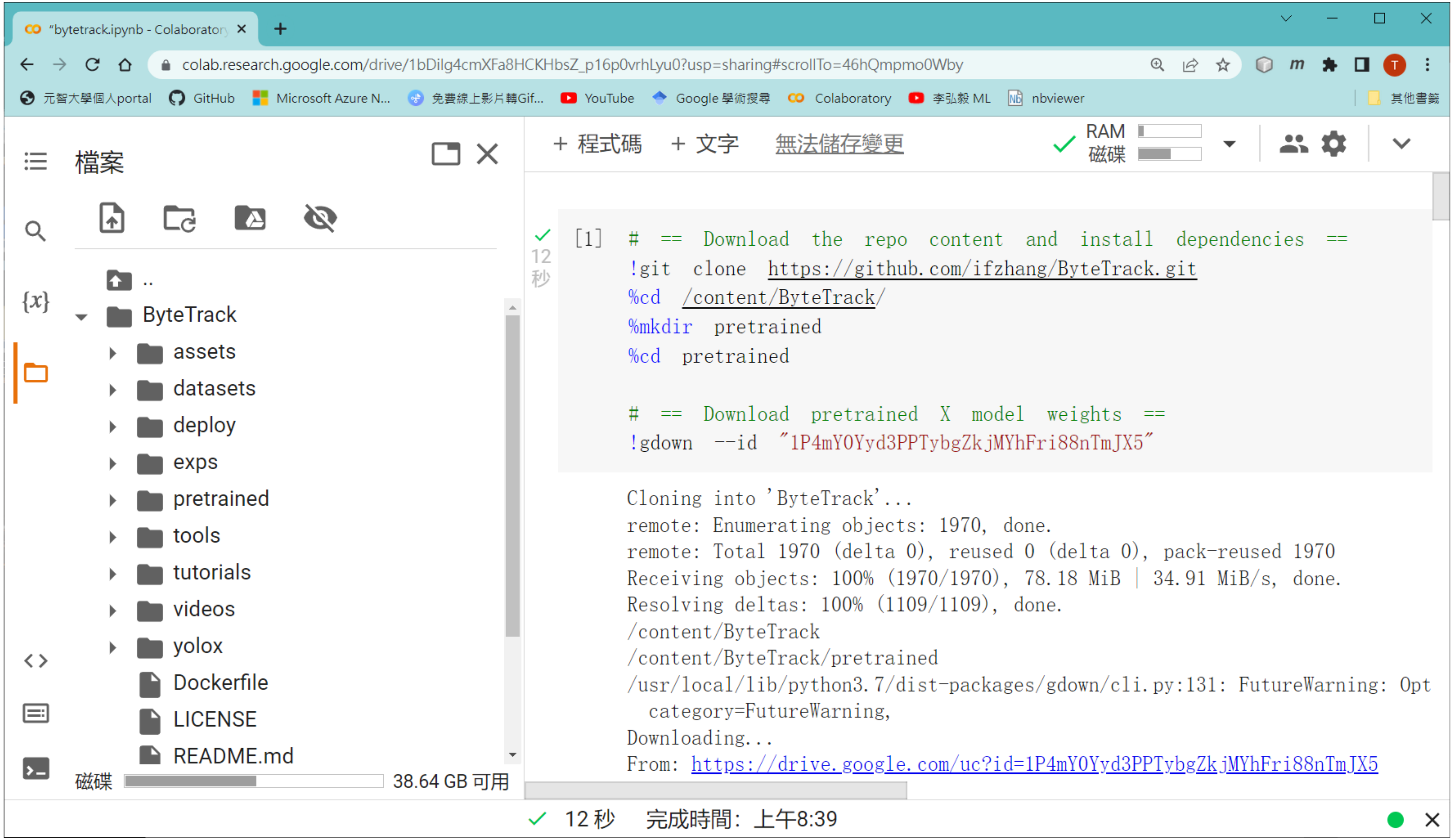
Multiple Object tracking



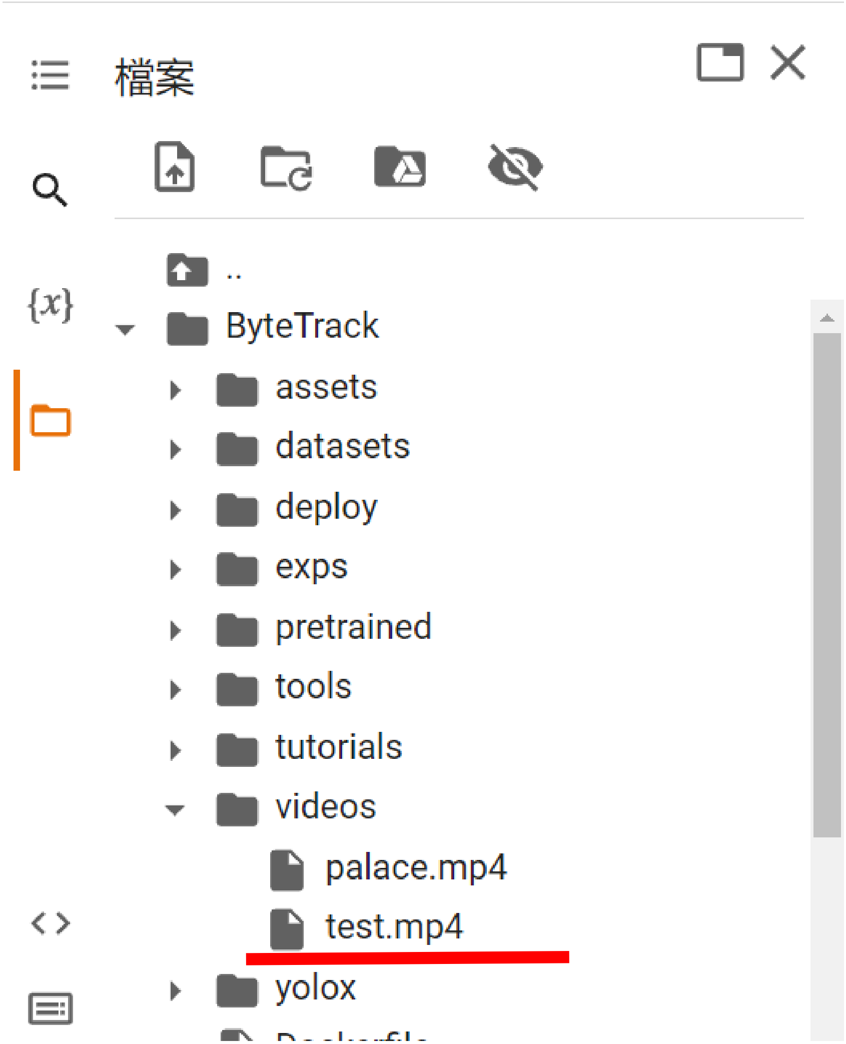
Class practice

ByteTrack.ipynb

Run first cell and take a look at ByteTrack folders



Upload our own video to the videos folder



Run object tracking on your own video

Runs very slow on Colab! Be patient

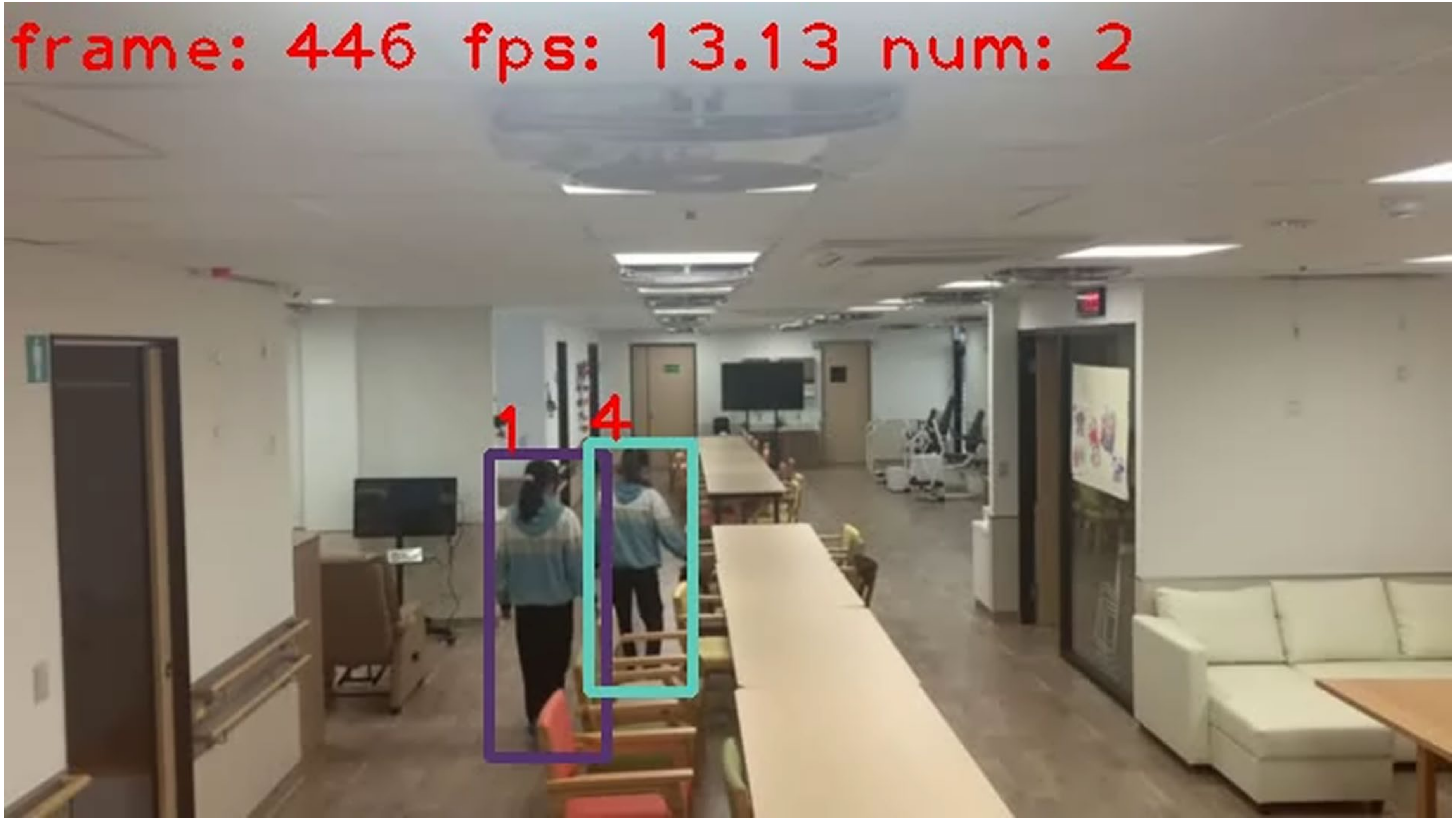
```
[11] # run inference demo (can be slow on colab). The cell output is deflected to the file 'lo
     %cd /content/ByteTrack
     !python3 tools/demo_track.py video --path="./videos/test.mp4" -f exps/example/mot/yolox_x_mix_det.py
```

```
--path="./videos/test.mp4"
```

- ByteTrack
  - YOLOX_outputs
    - yolox_x_mix_det
      - track_vis
        - 2022_06_03_02_13_11
          - test.mp4
        - 2022_06_03_02_13_11.txt

When finished, download the results.

Tracking results



Multiple object tracking

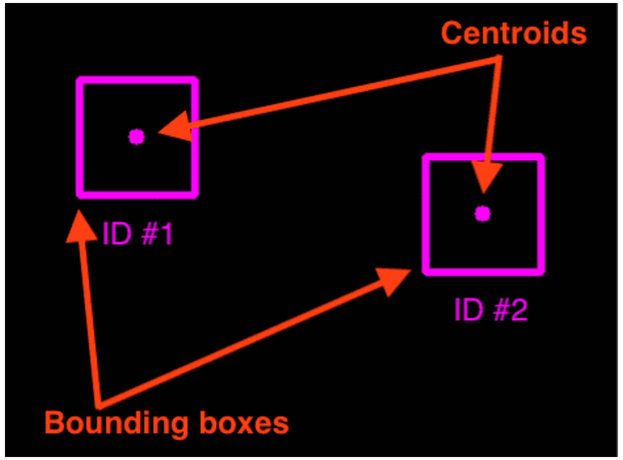
- Object tracking includes: 1) object detection, and 2) tracker

Frame t Frame t+1

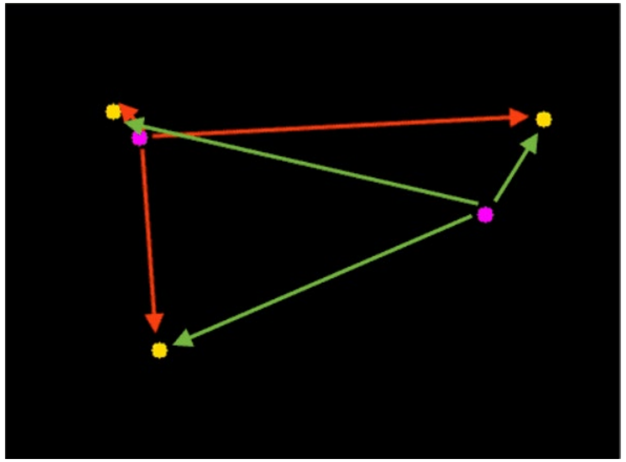
https://peaceful0907.medium.com/%E5%88%9D%E6%8E%A2%E7%89%A9%E4%BB%B6%E8%BF%BD%E8%B9%A4-multiple-object-tracking-mot-4f1b42e959f9

General tracker

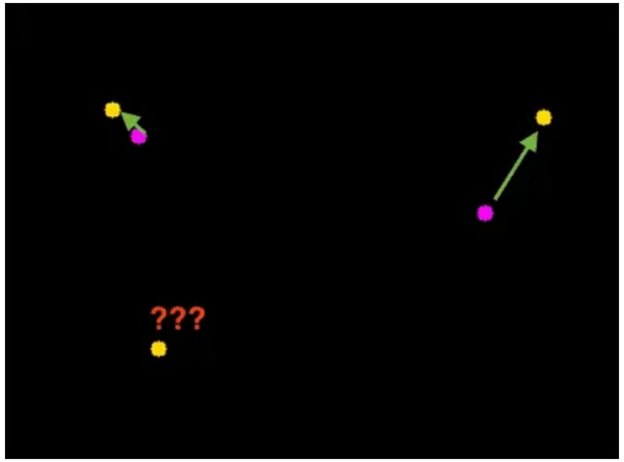
Accept bbox coordinates and compute centroids



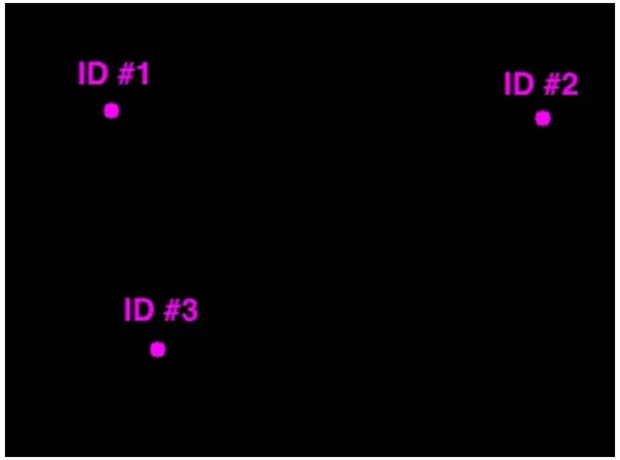
Compute Euclidean dist between new bboxes and existing objects



Update (x, y) coordinates of existing objects



Register new objects



- Two parameters: 1) max disappear frame, and 2) max distance

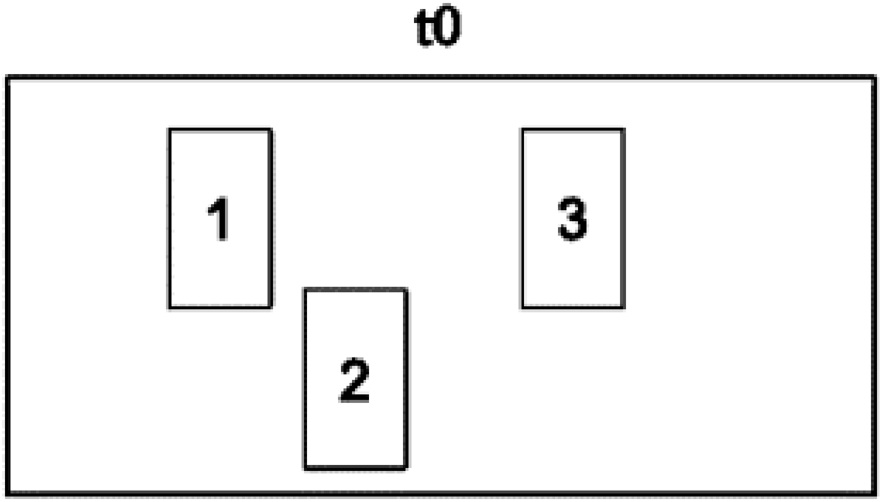
https://pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

SORT (Simple on-line real-time tracking)

- Kalman Filter – predict future positions based on current position.
- Hungarian algorithm (匈牙利演算法) – associate an obstacle from one frame to another, based on a score, e.g., IOU, shape score, convolution cost.

Computer Vision for tracking. In Computer Vision, one of the most... | by Jeremy Cohen | Medium

The Kalman filter – Initialization



For each box, we initialize Kalman Matrices with coordinates of the bounding boxes.

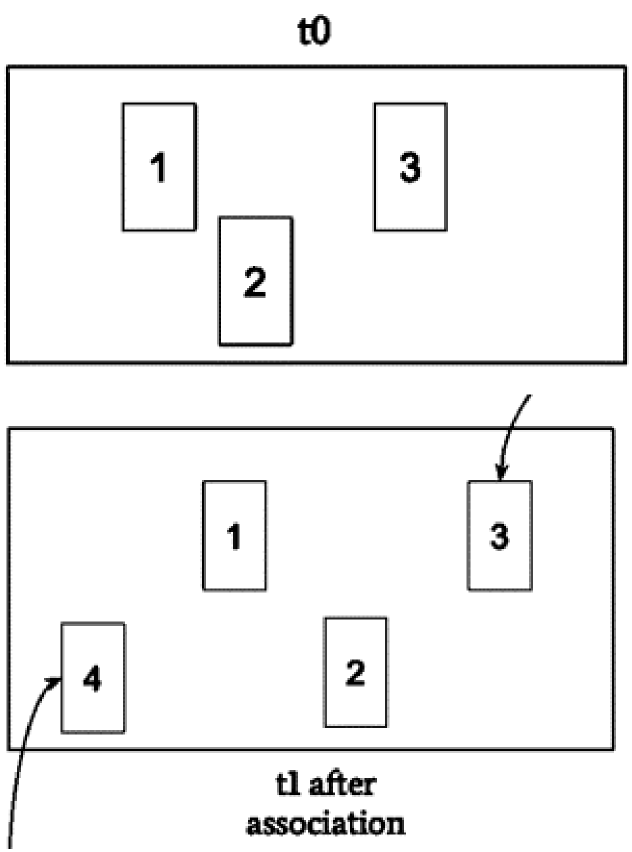
$$X = \begin{bmatrix} c_x & c_y & w & h & v_x & v_y & v_w & v_h \end{bmatrix}$$

$$P = \begin{bmatrix} 10 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 10 \end{bmatrix}$$

[Computer Vision for tracking. In Computer Vision, one of the most… | by Jeremy Cohen | Medium]

Prediction

For existed bounding boxes, we predict the actual bounding boxes at time t1 from the bounding boxes at time t0 and then update our prediction with the measurement at time t1.



$$\underbrace{\begin{pmatrix} cx \\ cy \\ w \\ h \\ vx \\ vy \\ vw \\ vh \end{pmatrix}_{t+1}}_{x'} \underbrace{=}_{=} \underbrace{\begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & \mathbf{dt} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{dt} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{dt} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{dt} \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{pmatrix}}_{F} \cdot \underbrace{\begin{pmatrix} cx \\ cy \\ w \\ h \\ vx \\ vy \\ vw \\ vh \end{pmatrix}_{t}}_{x}$$

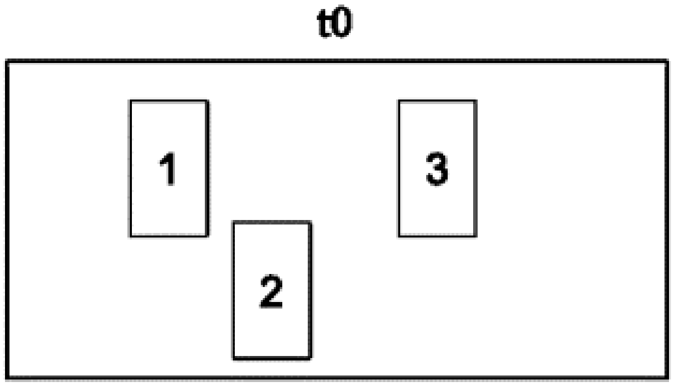
For new box, we initialize Kalman Matrices with coordinates of the bounding boxes.

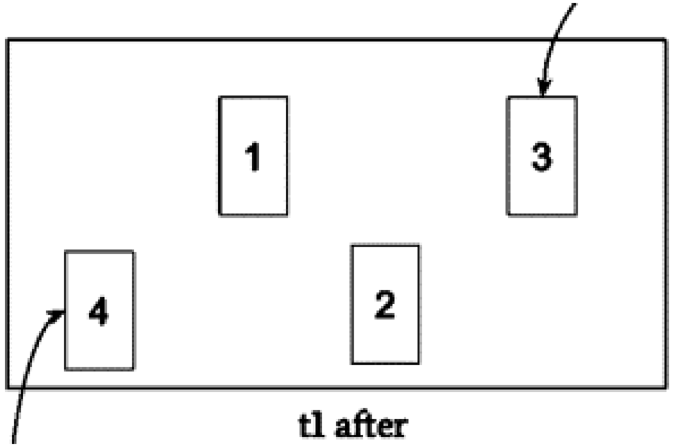
$$x' = Fx + u$$

$$P' = FPF^T + Q$$

[Computer Vision for tracking. In Computer Vision, one of the most… | by Jeremy Cohen | Medium]

Update





$$x' = Fx + u$$

$$P' = FPF^T + Q$$

$$y = z - Hx'$$

$$Z = \begin{bmatrix} c_x & c_y & w & h \end{bmatrix}$$

$$H = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{matrix}$$

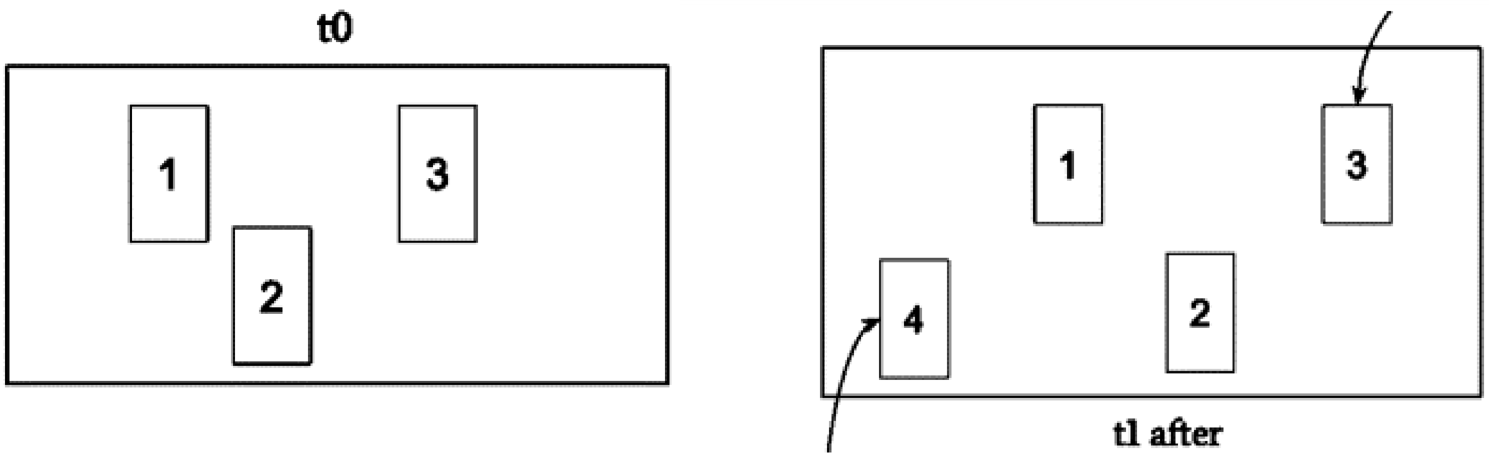
$$S = HP'H^T + R$$

$$R = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{matrix}$$

$$K = P'H^TS^{-1}$$

[Computer Vision for tracking. In Computer Vision, one of the most… | by Jeremy Cohen | Medium]

Update



$$x' = Fx + u$$
$$P' = FPF^T + Q$$

$$y = z - Hx'$$

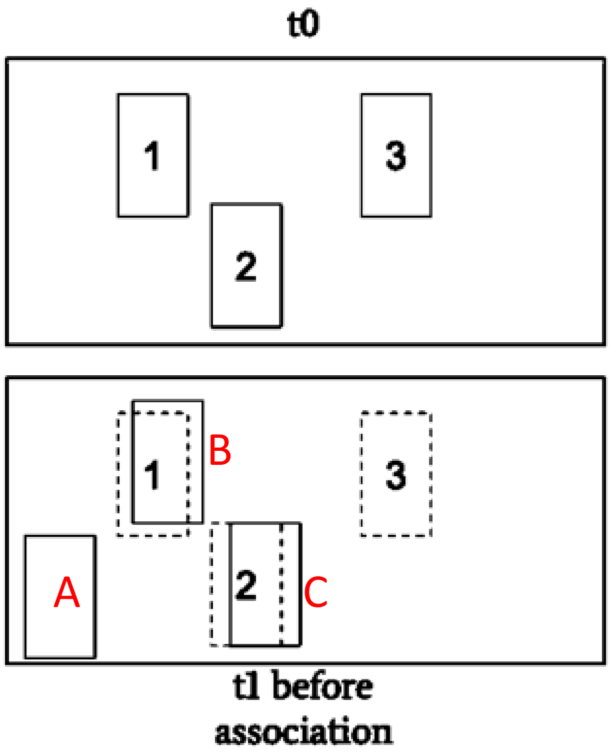
$$S = HP'H^T + R$$

$$K = P'H^TS^{-1}$$

$$x = x' + Ky$$
$$P = (I - KH)P'$$

[Computer Vision for tracking. In Computer Vision, one of the most… | by Jeremy Cohen | Medium]

Hungarian Algorithm

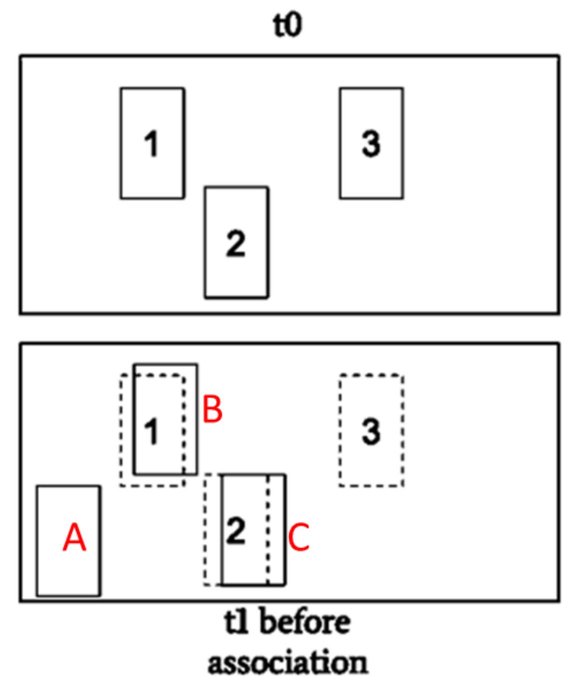


- We have two lists of boxes from object detection: a tracking list (t0: 1, 2, 3) and a detection list (t1: A, B, C).
- Go through tracking and detection list, and calculate IOU, (or shape, convolutional score). Store the IOU scores in a matrix.

Detection/Tracking	Tracking 1	Tracking 2	Tracking 3
Detection A	IOU = 0	IOU = 0	IOU = 0
Detection B	IOU = 0.56	IOU = 0	IOU = 0
Detection C	IOU = 0	IOU = 0.77	IOU = 0

Computer Vision for tracking. In Computer Vision, one of the most… | by Jeremy Cohen | Medium

Hungarian Algorithm



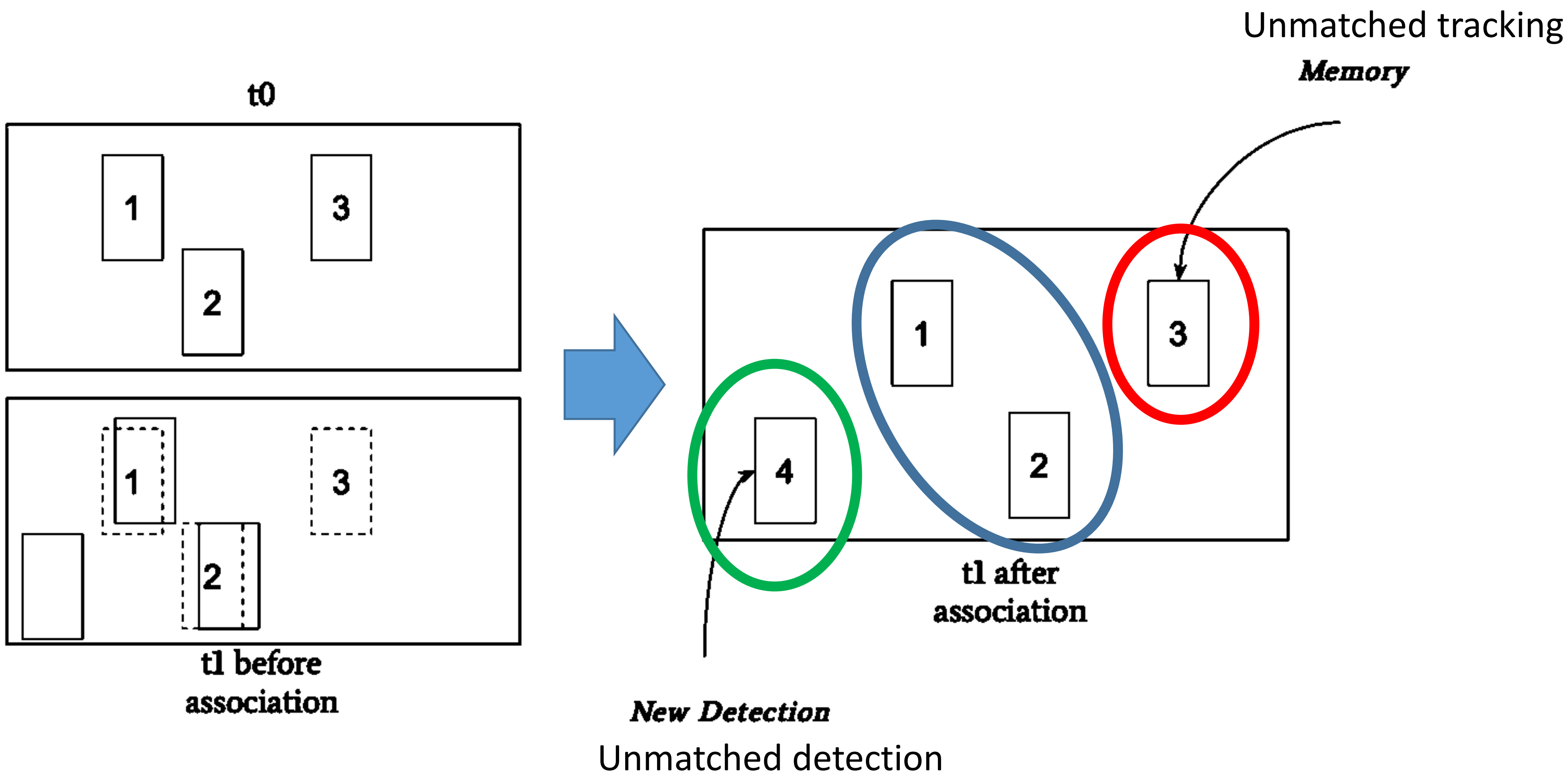
The next thing is to call a sklearn function called linear_assignment() that implements the Hungarian Algorithm. This algorithm uses bipartite graph (graph theory) to find for each detection, the lowest tracking value in the matrix. We can then check the values missing in our Hungarian Matrix and consider them as unmatched detections, or unmatched tracking.

Detection/Tracking	Tracking 1	Tracking 2	Tracking 3
Detection A	IOU = 0	IOU = 0	IOU = 0
Detection B	IOU = 0.56	IOU = 0	IOU = 0
Detection C	IOU = 0	IOU = 0.77	IOU = 0

ID tracking	ID Detection	
	A	Unmatched detection
1	B	
2	C	
3		Unmatched tracking

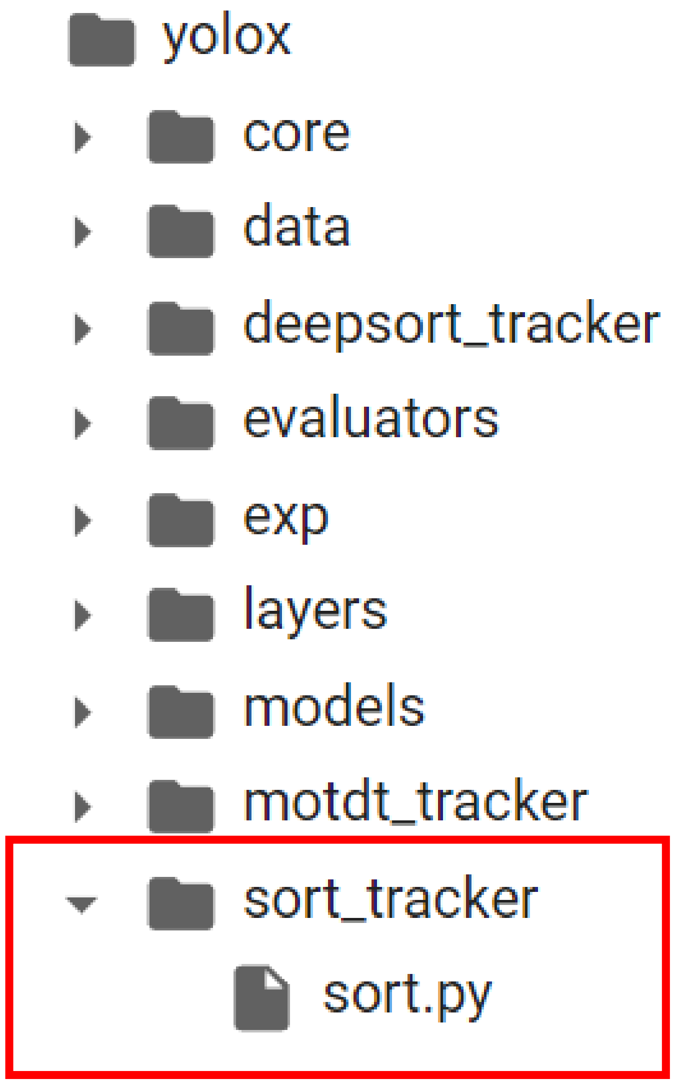
Computer Vision for tracking. In Computer Vision, one of the most… | by Jeremy Cohen | Medium

Hungarian Algorithm



[Computer Vision for tracking. In Computer Vision, one of the most… | by Jeremy Cohen | Medium]

Download sort.py and open with Spider



Class practice – Initialize F, P, H, R, Q

```
.kf = KalmanFilter(dim_x=7, dim_z=4)
.kf.F = np.array([[1,0,0,0,1,0,0],[0,1,0
.kf.H = np.array([[1,0,0,0,0,0,0],[0,1,0

.kf.R[2:,2:] *= 10.
.kf.P[4:,4:] *= 1000. #give high uncerta
.kf.P *= 10.
.kf.Q[-1,-1] *= 0.01
.kf.Q[4:,4:] *= 0.01

.kf.x[:4] = convert_bbox_to_z(bbox)
.time_since_update = 0
.id = KalmanBoxTracker.count
anBoxTracker.count += 1
.history = []
.hits = 0
.hit_streak = 0
.age = 0
```

$$x' = Fx + u$$

$$P' = FPF^T + Q$$

$$y = z - Hx'$$

$$S = HP'H^T + R$$

$$K = P'H^TS^{-1}$$

$$x = x' + Ky$$

$$P = (I - KH)P'$$

Class practice – SORT tracking

```
class Sort(object):
  def __init__(self, det_thresh, max_age=30, min_hits=3, iou_threshold=0.3):
```

```
    for t, trk in enumerate(trks):
      pos = self.trackers[t].predict()[0]
      trk[:] = [pos[0], pos[1], pos[2], pos[3], 0]
      if np.any(np.isnan(pos)):
        to_del.append(t)
```

$$x' = Fx + u$$

$$P' = FPF^T + Q$$

```
matched, unmatched_dets, unmatched_trks = associate_detections_to_trackers(dets, trks,
```

```
for m in matched:
  self.trackers[m[1]].update(dets[m[0], :])
```

$$y = z - Hx'$$

$$S = HP'H^T + R$$

$$K = P'H^T S^{-1}$$

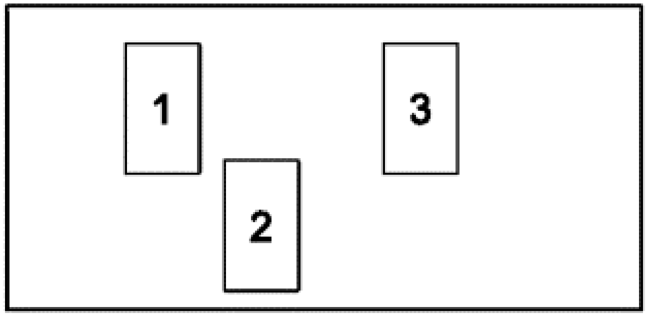
$$x = x' + Ky$$

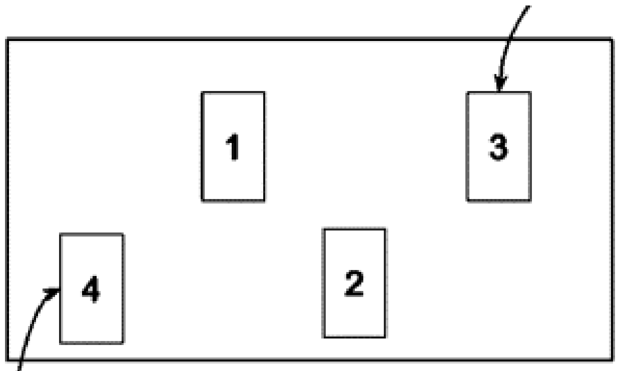
$$P = (I - KH)P'$$

Class practice – Prediction

```python
def predict(self):
  """
  Advances the state vector and returns
  """
  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]
```

t0





t1 after association

$$x' = Fx + u$$

$$P' = FPF^T + Q$$

Class practice – Update

```python
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))
```

$$y = z - Hx'$$

$$S = HP'H^T + R$$

$$K = P'H^T S^{-1}$$

$$x = x' + Ky$$

$$P = (I - KH)P'$$

Class practice – Hungarian Algorithm

```python
def associate_detections_to_trackers(detections,trackers,iou_threshold = 0.3):
  """
  Assigns detections to tracked object (both represented as bounding boxes)
  Returns 3 lists of matches, unmatched_detections and unmatched_trackers
  """
```

```python
matched_indices = linear_assignment(-iou_matrix)
```

```python
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)))
```

ID tracking	ID Detection	
	A	Unmatched detection
1	B	
2	C	
3		Unmatched tracking

ByteTrack

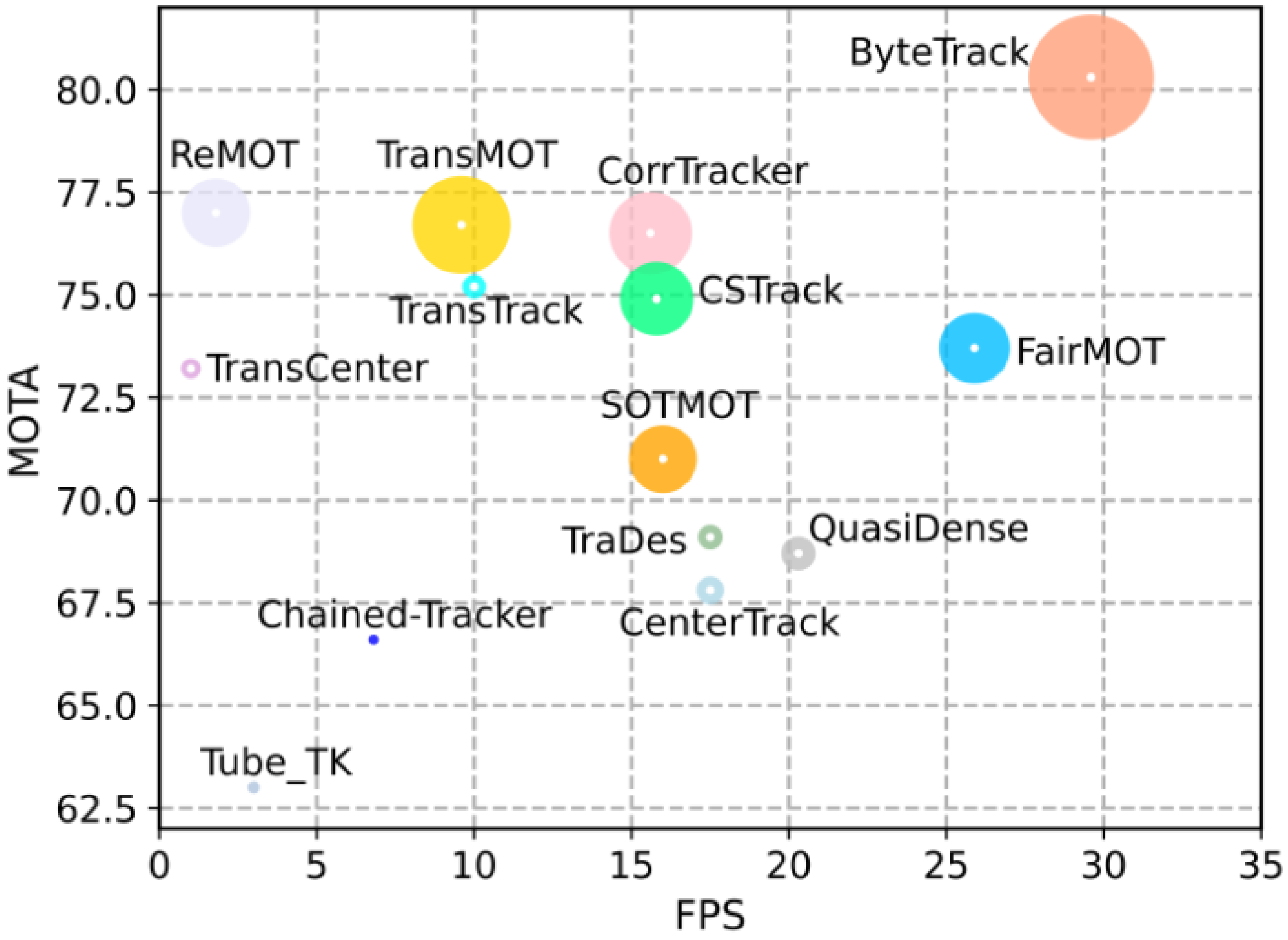
ByteTrack: Multi-Object Tracking by Associating Every Detection Box

Yifu Zhang[1], Peize Sun[2], Yi Jiang[3], Dongdong Yu[3], Fucheng Weng[1],
Zehuan Yuan[3], Ping Luo[2], Wenyu Liu[1], Xinggang Wang[1†]

[1]Huazhong University of Science and Technology [2]The University of Hong Kong [3]ByteDance Inc.

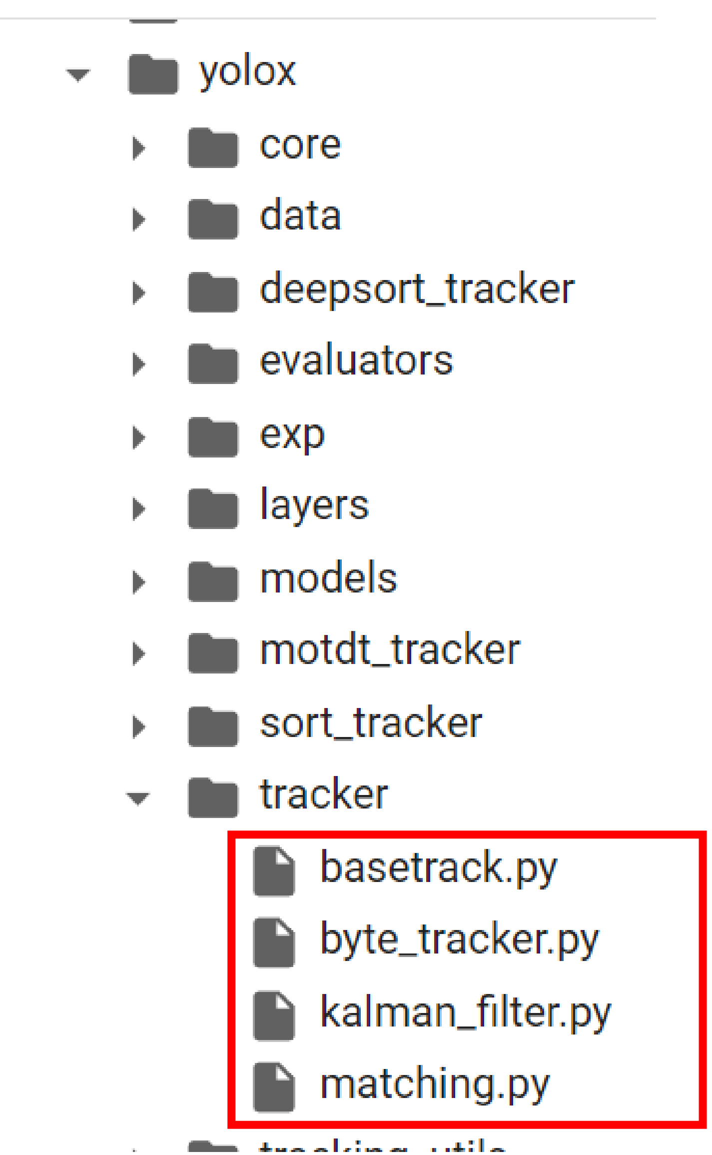
https://arxiv.org/pdf/2110.06864.pdf

ByteTrack



https://arxiv.org/pdf/2110.06864.pdf

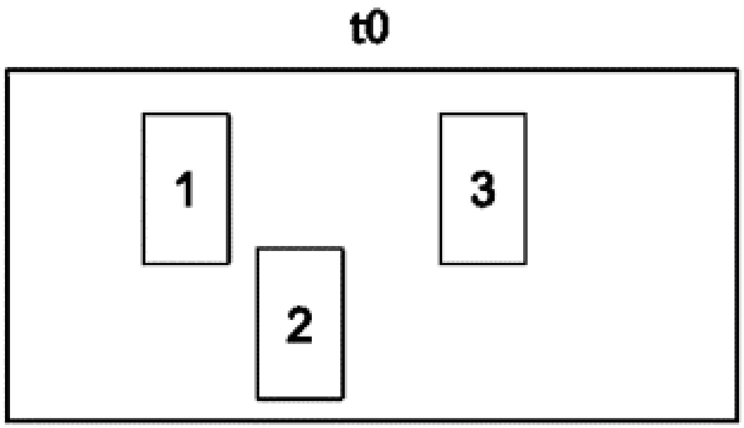
Download kalman filter, matching, bsetrack



Class practice – Initialization

```
mean_pos = measurement
mean_vel = np.zeros_like(mean_pos)
mean = np.r_[mean_pos, mean_vel]

std = [
    2 * self._std_weight_position * measurement[3],
    2 * self._std_weight_position * measurement[3],
    1e-2,
    2 * self._std_weight_position * measurement[3],
    10 * self._std_weight_velocity * measurement[3],
    10 * self._std_weight_velocity * measurement[3],
    1e-5,
    10 * self._std_weight_velocity * measurement[3]]
covariance = np.diag(np.square(std))
return mean, covariance
```



$$X = \begin{bmatrix} c_x & c_y & w & h & v_x & v_y & v_w & v_h \end{bmatrix}$$

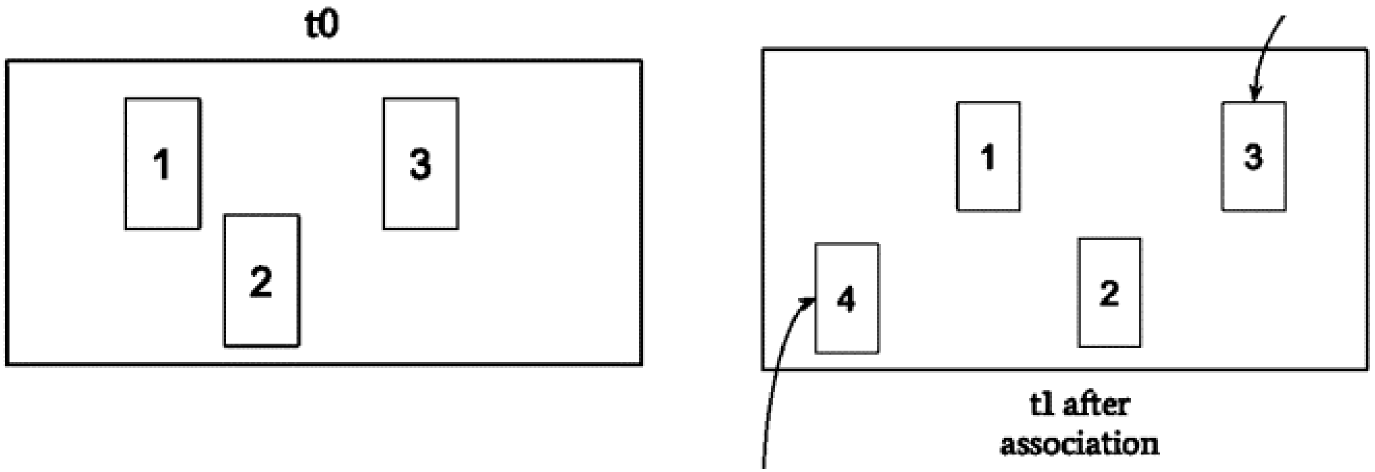
$$P = \begin{bmatrix} 10 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 10 \end{bmatrix}$$

Class practice – Prediction

```python
std_pos = [
    self._std_weight_position * mean[3],
    self._std_weight_position * mean[3],
    1e-2,
    self._std_weight_position * mean[3]]
std_vel = [
    self._std_weight_velocity * mean[3],
    self._std_weight_velocity * mean[3],
    1e-5,
    self._std_weight_velocity * mean[3]]
motion_cov = np.diag(np.square(np.r_[std_pos, std_vel]))

#mean = np.dot(self._motion_mat, mean)
mean = np.dot(mean, self._motion_mat.T)
covariance = np.linalg.multi_dot((
    self._motion_mat, covariance, self._motion_mat.T)) + motion_cov

return mean, covariance
```



$$x' = Fx + u$$

$$P' = FPF^T + Q$$

Class practice – Motion matrix

```python
ndim, dt = 4, 1.

# Create Kalman filter model matrices.
self._motion_mat = np.eye(2 * ndim, 2 * ndim)
for i in range(ndim):
    self._motion_mat[i, ndim + i] = dt
self._update_mat = np.eye(ndim, 2 * ndim)

# Motion and observation uncertainty are chose
# state estimate. These weights control the an
# the model. This is a bit hacky.
self._std_weight_position = 1. / 20
self._std_weight_velocity = 1. / 160
```

$$
\underbrace{\begin{pmatrix} cx \\ cy \\ w \\ h \\ vx \\ vy \\ vw \\ vh \end{pmatrix}_{t+1}}_{x'}
\underbrace{=}_{=}
\underbrace{\begin{pmatrix}
1 & 0 & 0 & 0 & dt & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & dt & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & dt & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & dt \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}}_{F}
\cdot
\underbrace{\begin{pmatrix} cx \\ cy \\ w \\ h \\ vx \\ vy \\ vw \\ vh \end{pmatrix}_{t}}_{x}
$$

$$x' = Fx + u$$

$$P' = FPF^T + Q$$

Class practice – Update

```
projected_mean, projected_cov = self.project(mean, covariance)

chol_factor, lower = scipy.linalg.cho_factor(
    projected_cov, lower=True, check_finite=False)
kalman_gain = scipy.linalg.cho_solve(
    (chol_factor, lower), np.dot(covariance, self._update_mat.T).T,
    check_finite=False).T
innovation = measurement - projected_mean

new_mean = mean + np.dot(innovation, kalman_gain.T)
new_covariance = covariance - np.linalg.multi_dot((
    kalman_gain, projected_cov, kalman_gain.T))
return new_mean, new_covariance
```

$$y = z - Hx'$$

$$S = HP'H^T + R$$

$$K = P'H^T S^{-1}$$

$$x = x' + Ky$$

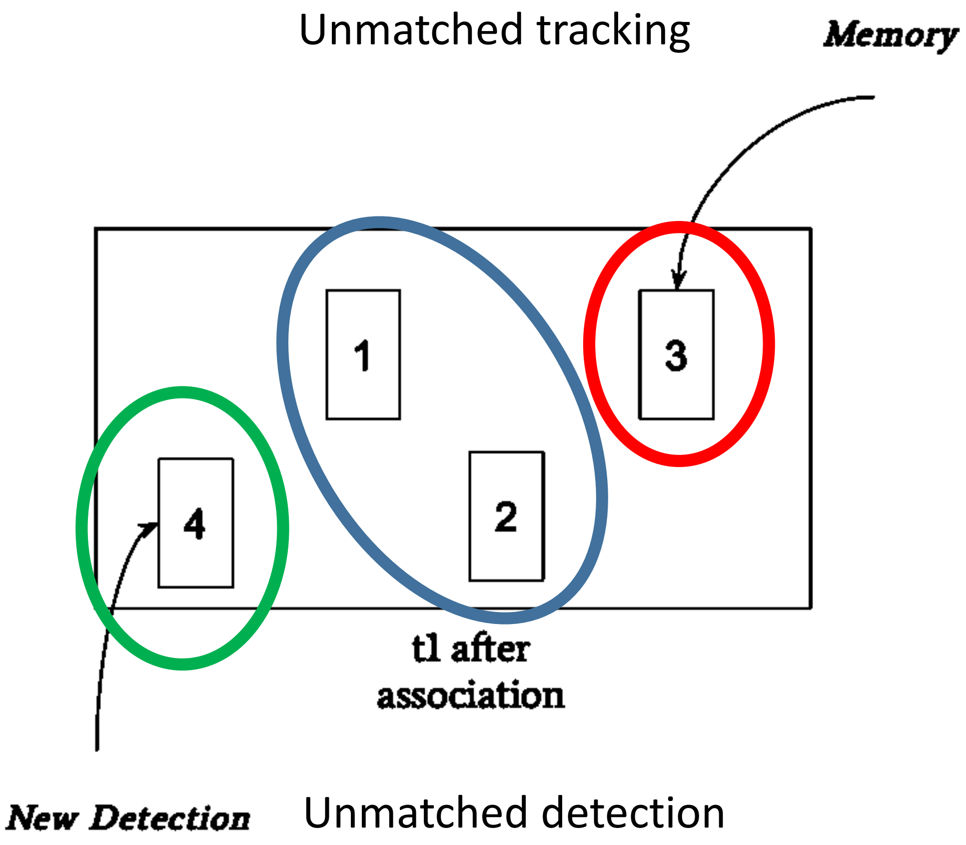
$$P = (I - KH)P'$$

Class practice – matching.py

```
: linear_assignment(cost_matrix, thresh):
  if cost_matrix.size == 0:
      return np.empty((0, 2), dtype=int), tuple(rar
  matches, unmatched_a, unmatched_b = [], [], []
  cost, x, y = lap.lapjv(cost_matrix, extend_cost=1
  for ix, mx in enumerate(x):
      if mx >= 0:
          matches.append([ix, mx])
  unmatched_a = np.where(x < 0)[0]
  unmatched_b = np.where(y < 0)[0]
  matches = np.asarray(matches)
  return matches, unmatched_a, unmatched_b
```

ID tracking	ID Detection	
	A	Unmatched detection
1	B	
2	C	
3		Unmatched tracking

Class practice – basetrack.py




```python
class TrackState(object):
    New = 0
    Tracked = 1
    Lost = 2
    Removed = 3
```

```python
def next_id():
    BaseTrack._count += 1
    return BaseTrack._count

def activate(self, *args):
    raise NotImplementedError

def predict(self):
    raise NotImplementedError

def update(self, *args, **kwargs):
    raise NotImplementedError

def mark_lost(self):
    self.state = TrackState.Lost

def mark_removed(self):
    self.state = TrackState.Removed
```

More reading about object tracking

Computer Vision for tracking https://thinkautonomous.medium.com/computer-vision-for-tracking-8220759eee85

Multiple object tracking
https://peaceful0907.medium.com/%E5%88%9D%E6%8E%A2%E7%89%A9%E4%BB%B6%E8%BF%BD%E8%B9%A4-multiple-object-tracking-mot-4f1b42e959f9

Deep Sort
https://zhuanlan.zhihu.com/p/90835266?fbclid=IwAR1PYRbAhfb1LauTrgF9Az9GyTGR6A3-Lcm_1MPZE4i_t0DQhn2NEs_oLWo

Multiple Object Tracking
https://alu2019.home.blog/2021/01/20/edge-ai-multiple-object-tracking-mot-duo-ge-wu-ti/