Movement detection and object tracking

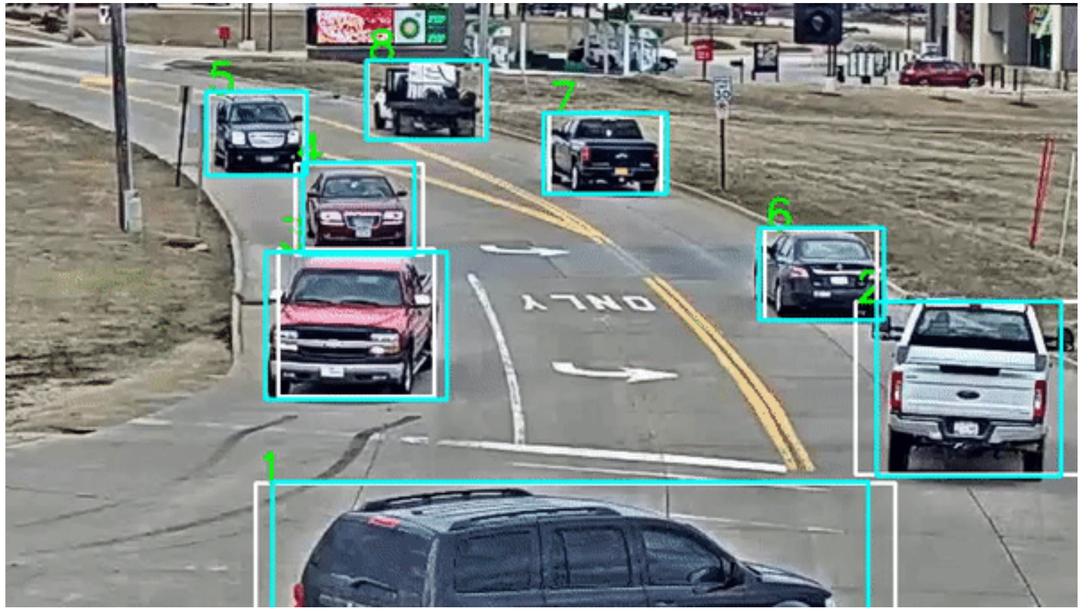
Computer Vision lecture
by Roman Polishchenko

Agenda

- Problem statement
- Basic approach
  - Optical flow
  - Lukas-Kanade algorithm
- Tracking-by-detection
  - Kalman filter
  - Hungarian algorithm
  - SORT
  - DeepSORT
- Tracking with network flows
- Your task

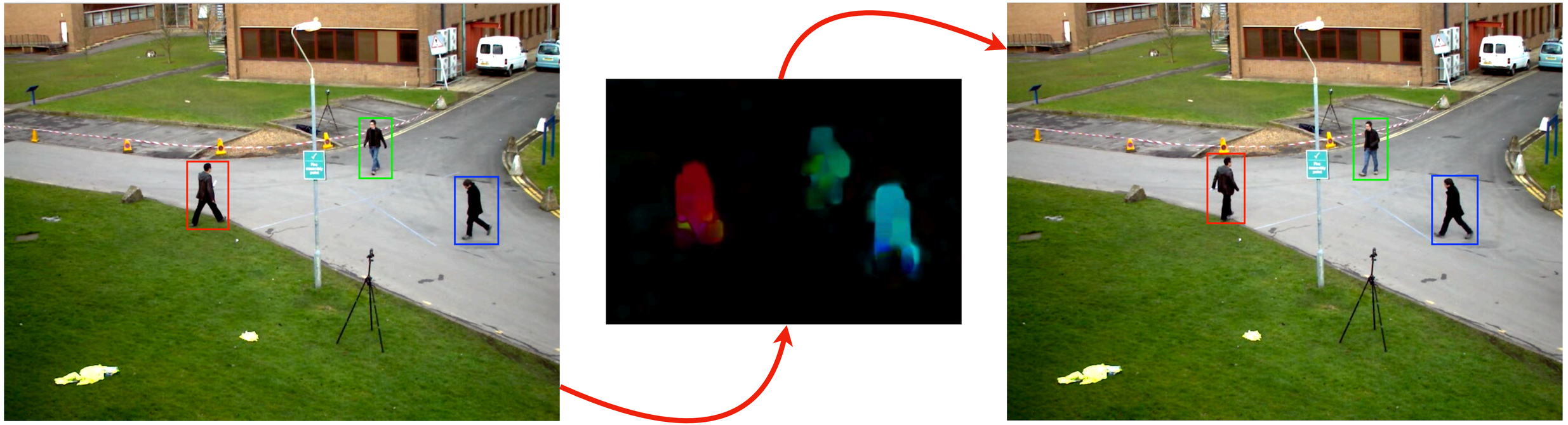
Problem statement

- Given a video, find out which parts of the image depict the same object(s) in different frames
- It can be a **single object** as shown on the left, but most of the time – it is needed to detect and perform tracking of **multiple objects**

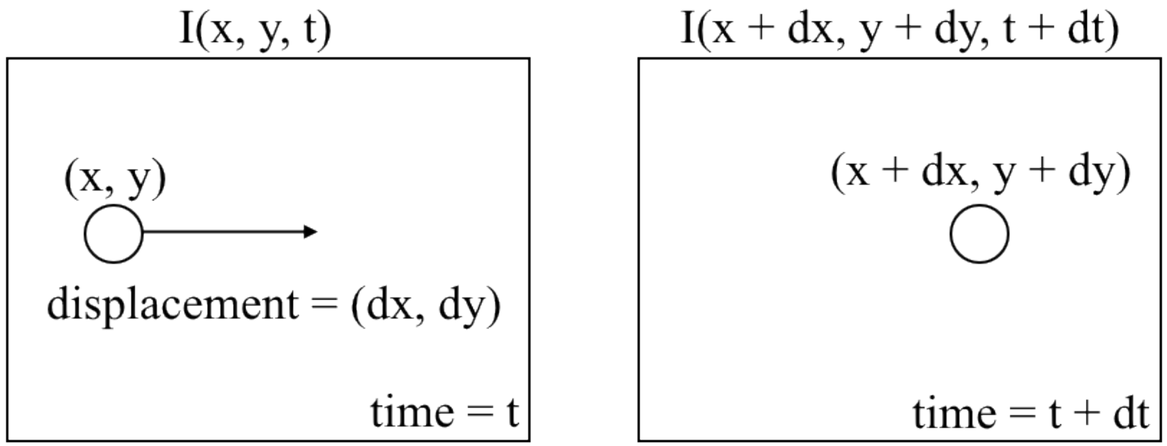


Basic approach (using movement detection)

- Camera is static
- Background is static
- Pre-defined region or bounding box to track



Optical flow: basics



$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$$

Assumption

Taylor Series Approximation

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + \dots$$

$$\Rightarrow \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0$$

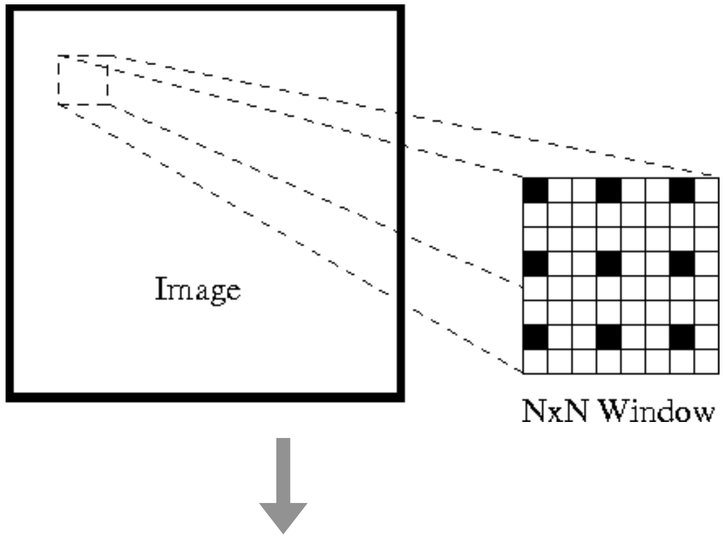
Divide by δt

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0$$

where (u, v) – movement vector

https://nanonets.com/blog/optical-flow/ https://docs.opencv.org/master/d4/dee/tutorial_optical_flow.html

Optical flow: Lukas-Kanade + pyramids

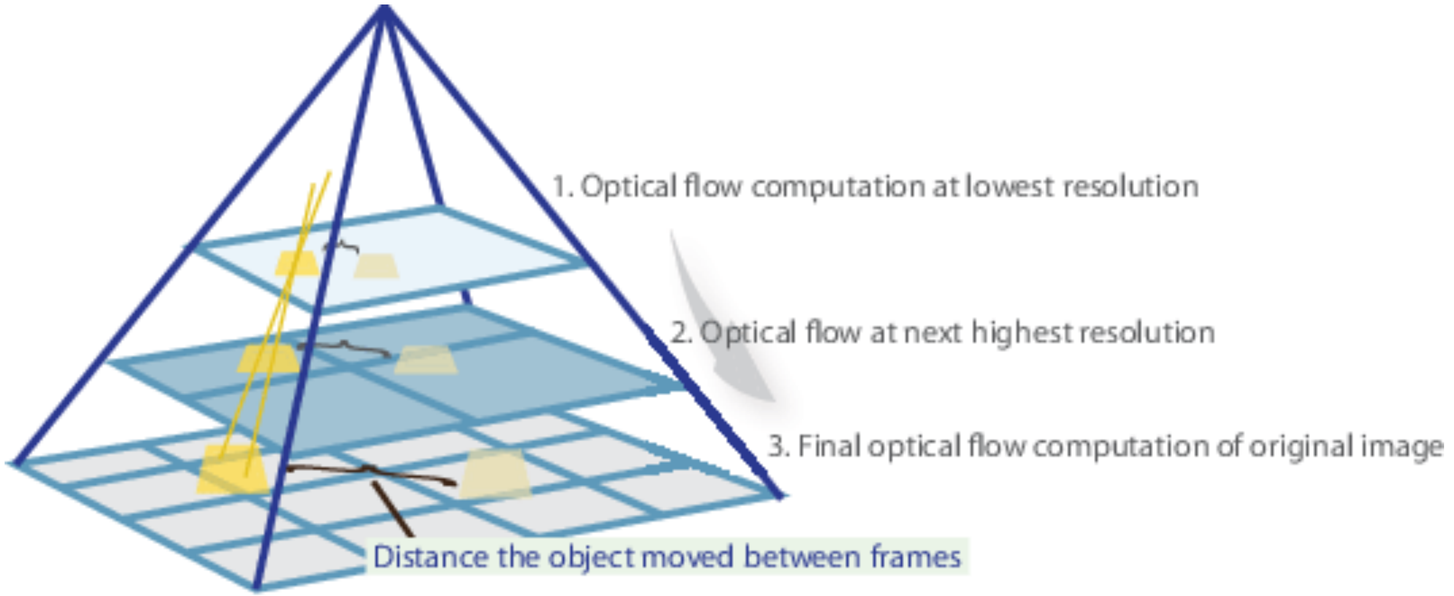


$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1)$$

$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$

$$\vdots$$

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n)$$



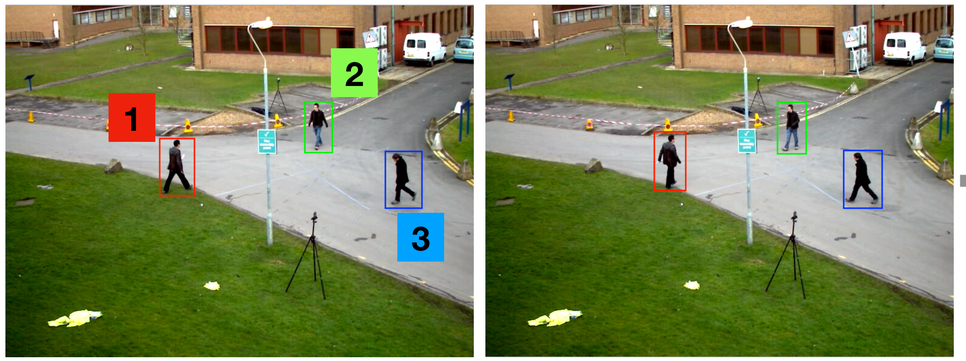
https://mathworld.wolfram.com/LeastSquaresFitting.html

http://robots.stanford.edu/cs223b04/algo_tracking.pdf

Tracking-by-detection

Performs in two steps:

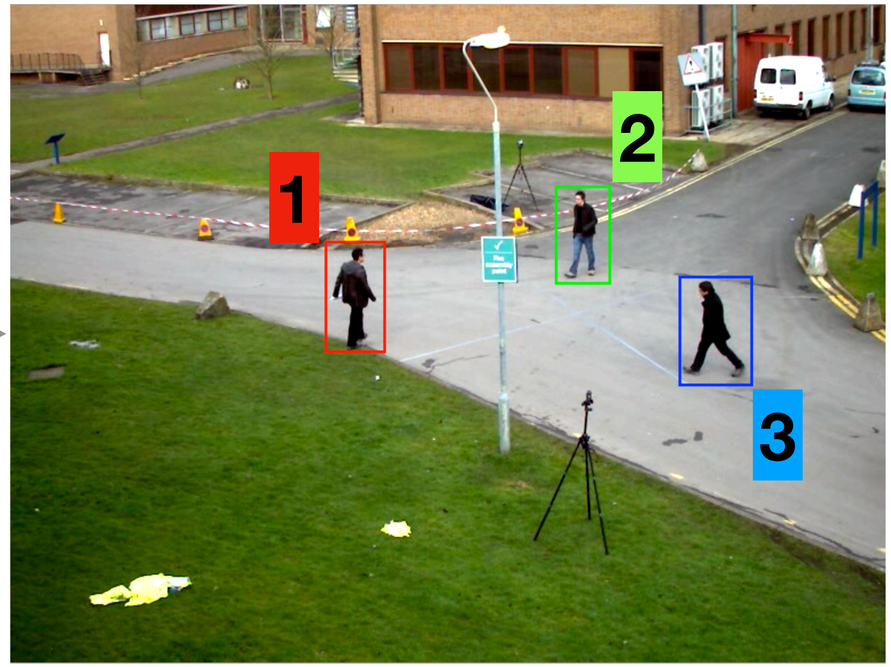
1. Detect object(s) on current frame.
2. Match previous and current detections.



t

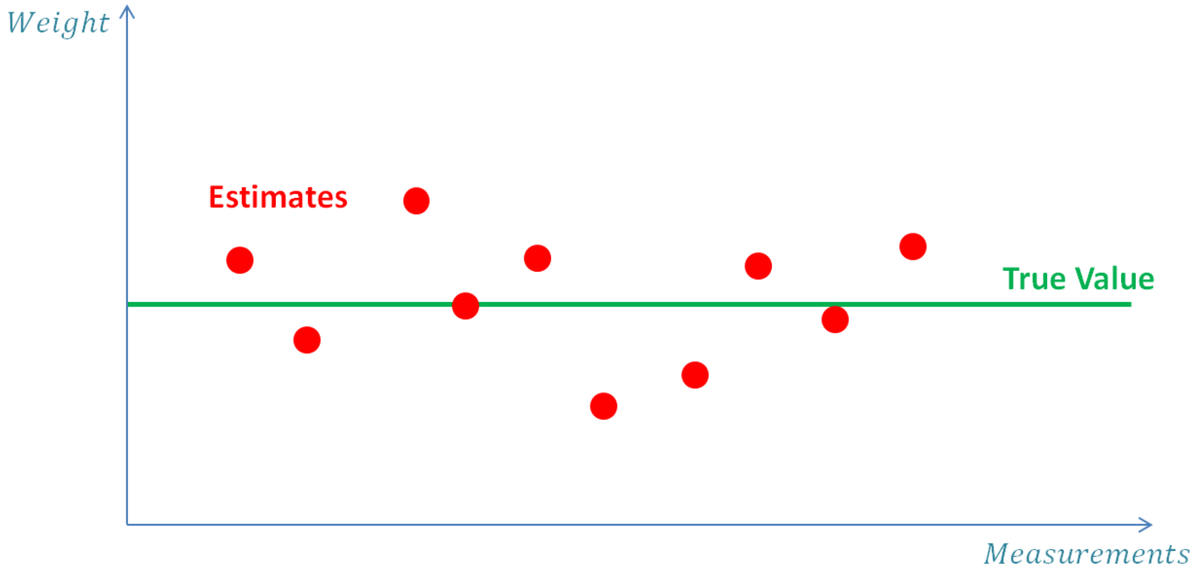
t+1

Matching algorithm



t+1

What's in black box? Kalman Filter



We got measurements:

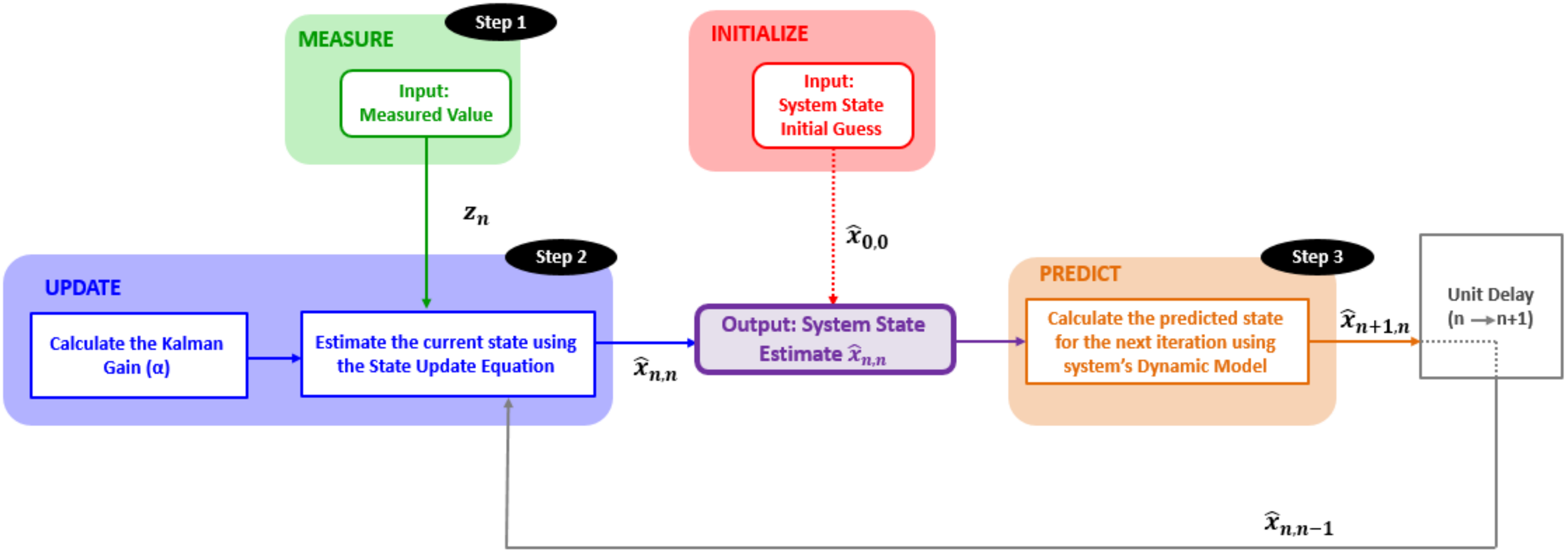
1. We want to estimate true value as accurate as possible
2. We don't want to save all previous values

State update equation

$$\text{The estimate of the current state} = \text{Predicted value of the current state} + \text{Factor} \times \left(\text{Measurement} - \text{Predicted value of the current state} \right)$$

https://www.kalmanfilter.net/alphabeta.html

What's in black box? Kalman Filter

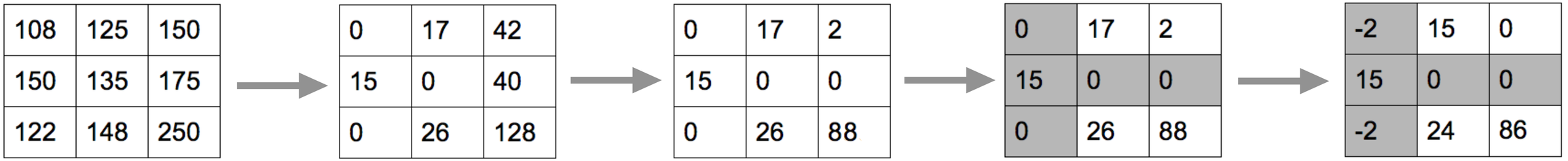


https://www.kalmanfilter.net/alphabeta.html

What’s in black box? Hungarian algorithm

The Hungarian Method:

1. Subtract the smallest entry in each row from all the other entries in the row. This will make the smallest entry in the row now equal to 0.
2. Subtract the smallest entry in each column from all the other entries in the column. This will make the smallest entry in the column now equal to 0.
3. Draw lines through the row and columns that have the 0 entries such that the fewest lines possible are drawn.
4. If there are n lines drawn, an optimal assignment of zeros is possible and the algorithm is finished. If the number of lines is less than n, then the optimal number of zeroes is not yet reached. Go to the next step.
5. Find the smallest entry not covered by any line. Subtract this entry from each row that isn’t crossed out, and then add it to each column that is crossed out. Then, go back to Step 3.

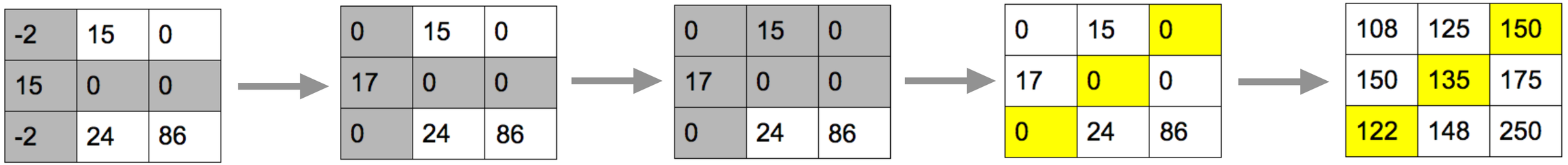


https://brilliant.org/wiki/hungarian-matching/#the-hungarian-algorithm-using-an-adjacency-matrix

What’s in black box? Hungarian algorithm

The Hungarian Method:

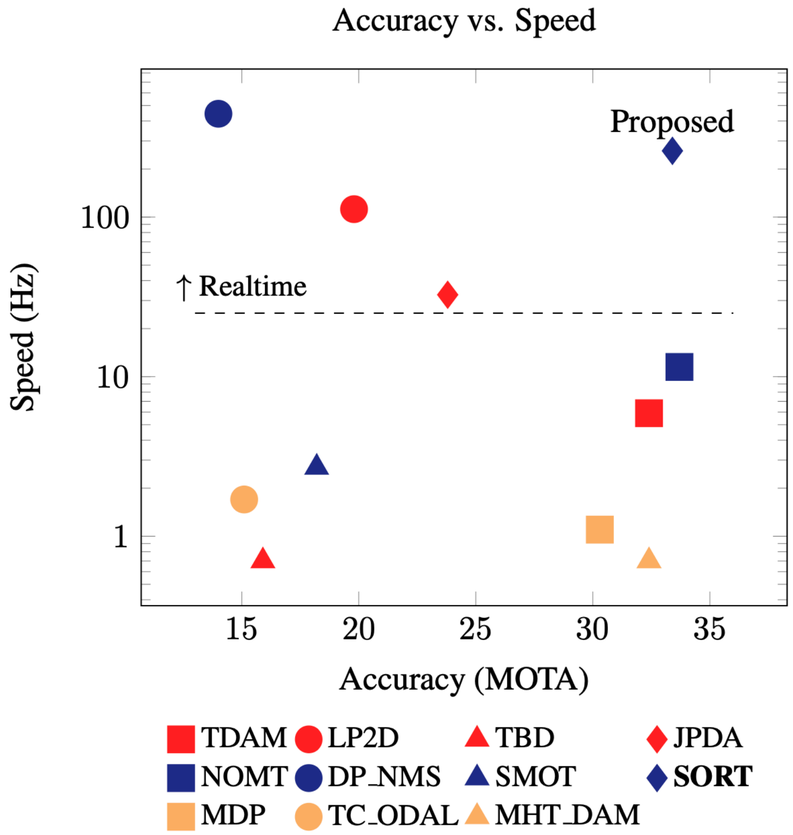
1. Subtract the smallest entry in each row from all the other entries in the row. This will make the smallest entry in the row now equal to 0.
2. Subtract the smallest entry in each column from all the other entries in the column. This will make the smallest entry in the column now equal to 0.
3. Draw lines through the row and columns that have the 0 entries such that the fewest lines possible are drawn.
4. If there are n lines drawn, an optimal assignment of zeros is possible and the algorithm is finished. If the number of lines is less than n, then the optimal number of zeroes is not yet reached. Go to the next step.
5. Find the smallest entry not covered by any line. Subtract this entry from each row that isn’t crossed out, and then add it to each column that is crossed out. Then, go back to Step 3.



https://brilliant.org/wiki/hungarian-matching/#the-hungarian-algorithm-using-an-adjacency-matrix

Black box == SORT (Simple Online and Realtime Tracking)

Compared to other algorithms in original paper in 2017:



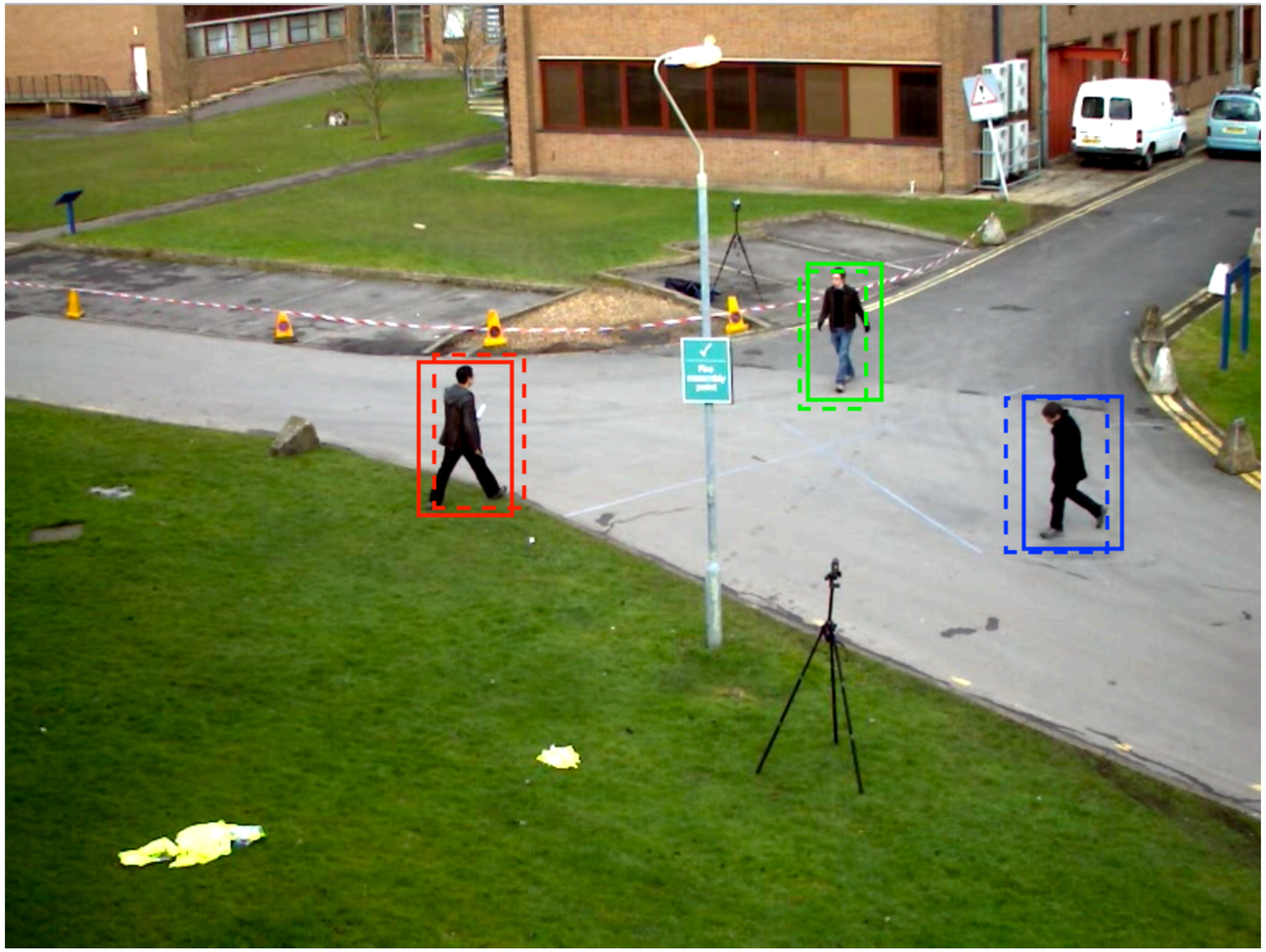
Object representation: $x = [u, v, s, t, \dot{u}, \dot{v}, \dot{s}]^T$

- u and v – horizontal and vertical pixel location of the centre of the target
- s and r – scale (area) and the aspect ratio of the target's bounding box
- $\dot{u}, \dot{v}, \dot{s}$ – respective velocities of the variables, that are solved optimally via a Kalman filter framework

- The assignment cost matrix is then computed as the **intersection-over-union** IOU distance between each detection and all predicted bounding boxes from the existing targets.
- A minimum IOU is imposed to reject assignments where the detection to target overlap is less than IOU_{min}.
- Tracks are initialised if they are detected for T_{init} frames.
- Tracks are terminated if they are not detected for T_{lost} frames.

https://arxiv.org/abs/1602.00763

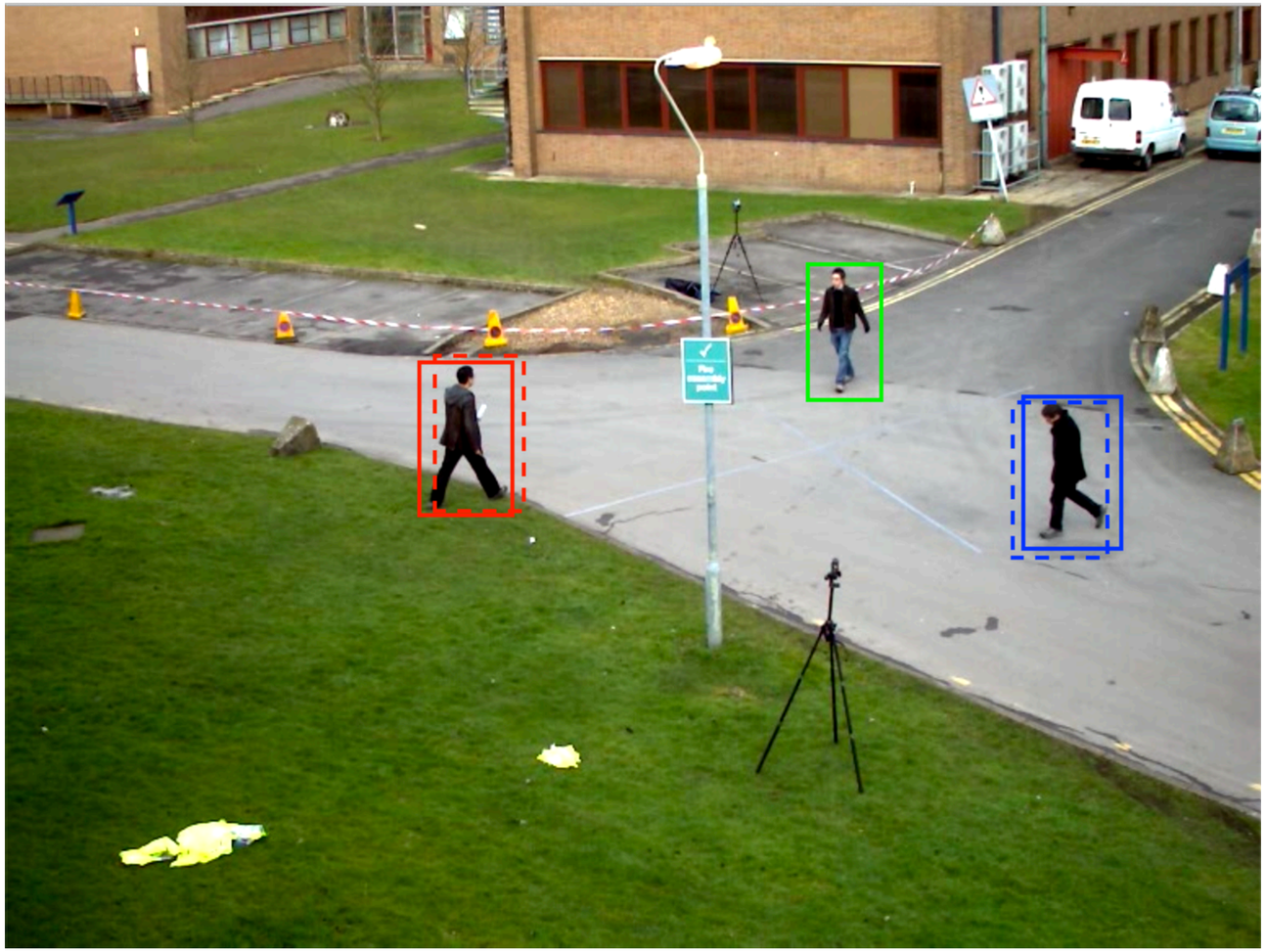
SORT (Simple Online and Realtime Tracking)



Example #1

IOU	Red Prediction	Green Prediction	Blue Prediction
Red Observation	0.93	0	0
Green Observation	0	0.88	0
Blue Observation	0	0	0.9

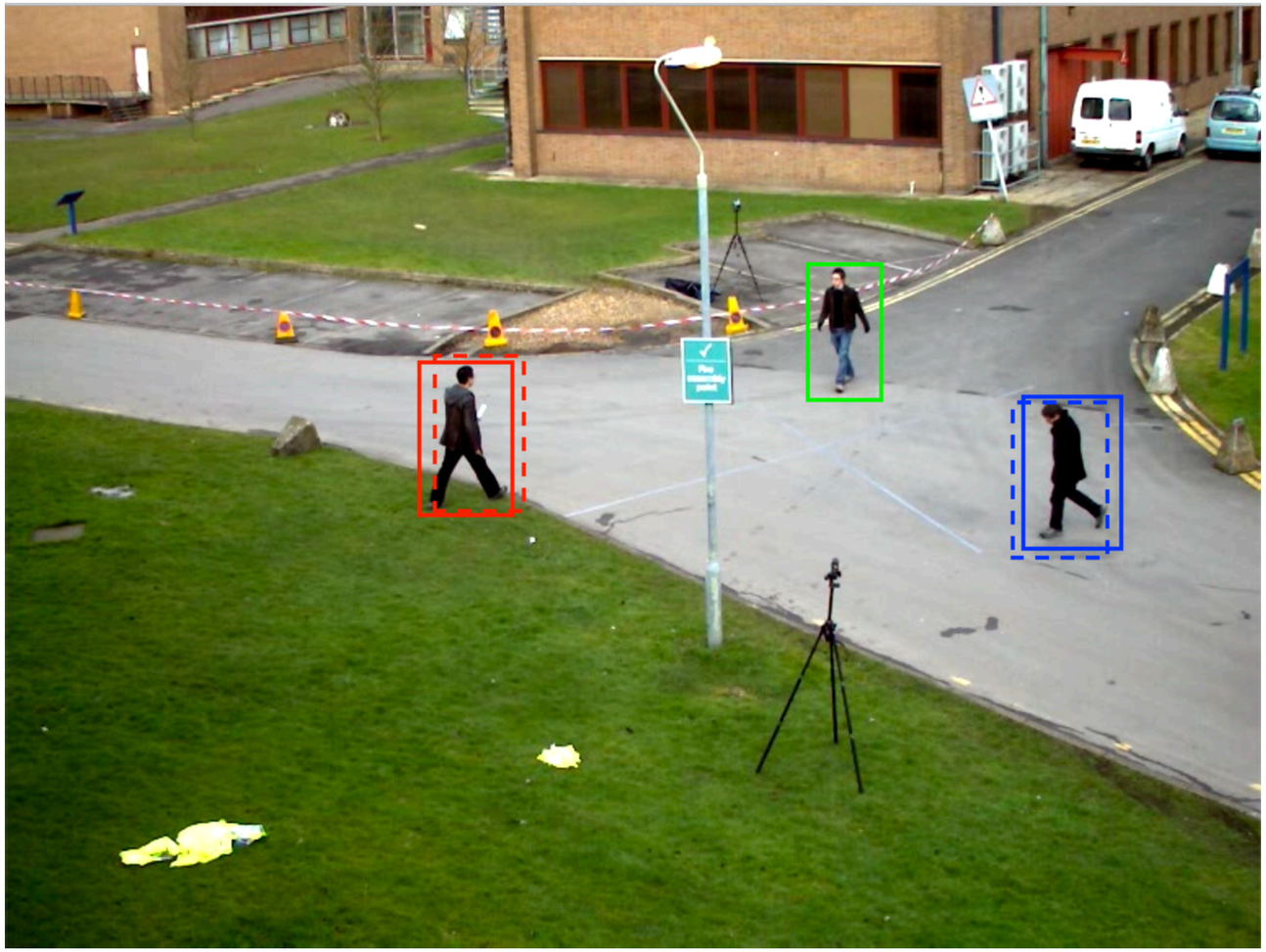
SORT (Simple Online and Realtime Tracking)



Example #2

IOU	Red Prediction	Green Prediction	Blue Prediction
Red Observation	0.93	0	0
Green Observation	0	0	0
Blue Observation	0	0	0.9

SORT (Simple Online and Realtime Tracking)



Example #2

IOU	Red Pred.	Green Pred.	Blue Pred.	Thresh for Red	Thresh for Green	Thresh for Blue
Red Observation	0.93	0	0	0.3	0	0
Green Observation	0	0	0	0	0.3	0
Blue Observation	0	0	0.9	0	0	0.3

Can we do better? Yep, DeepSORT

Idea: add neural network to obtain feature vector (‘“appearance descriptor”) of detected objects

$$D_{new} = \lambda * D_M + (1 - \lambda) * D_a$$

Mahalanobis distance
(or Euclidian distance, or IOU etc.)

Cosine similarity between
appearance vectors

$\vec{x} = (x_1, x_2, \ldots, x_N)^T$ – observation, $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_N)^T$ – mean,
S – **covariance matrix**

$$D_M = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}$$

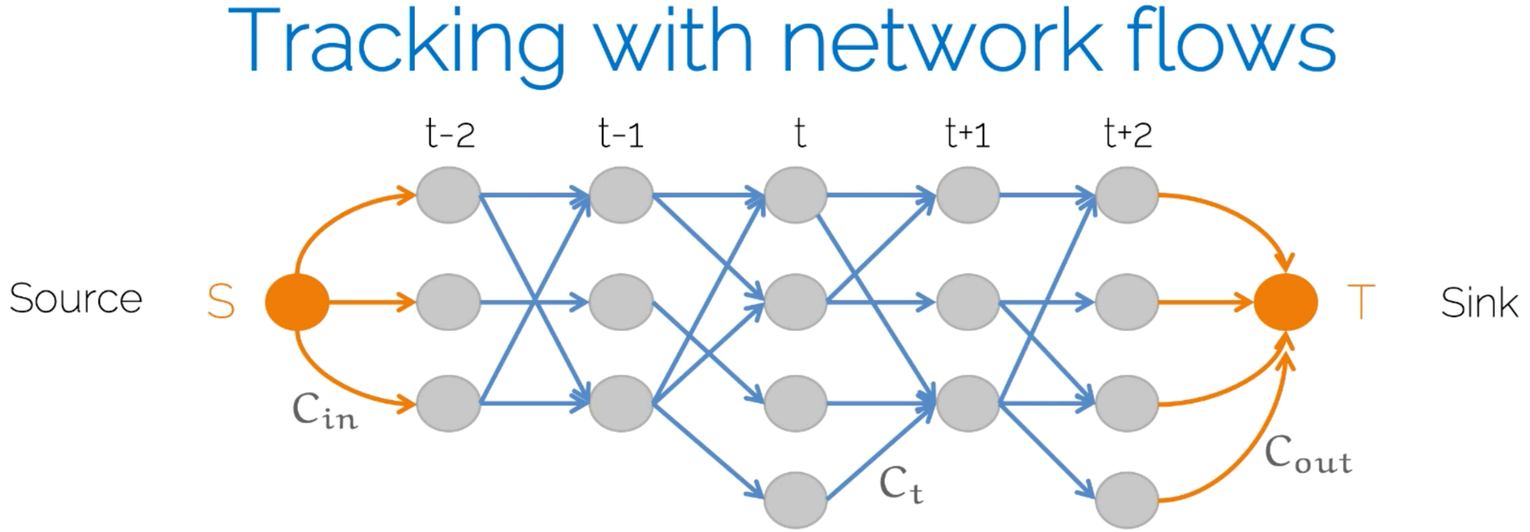
V_{last} – last computed appearance vector,
V_{new} – appearance vector of new observation

$$D_a = \frac{V_{last} \cdot V_{new}}{\| V_{last} \| \ \| V_{new} \|}$$

https://arxiv.org/abs/1703.07402

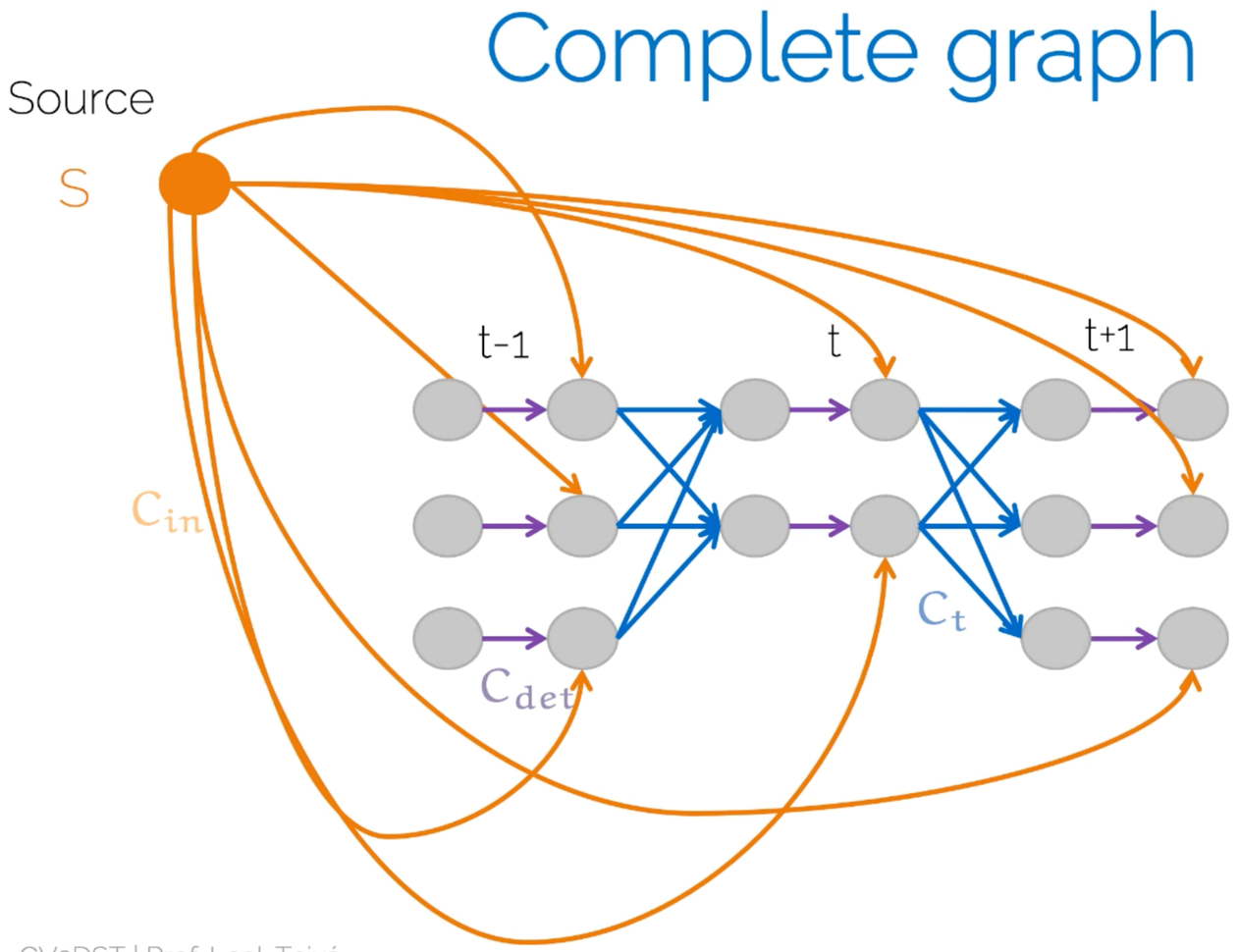
Tracking with network flows

~~Nobody does this, but~~ It looks fascinating!



https://www.youtube.com/watch?v=BR3Y5bAz5Dw&t

Tracking with network flows



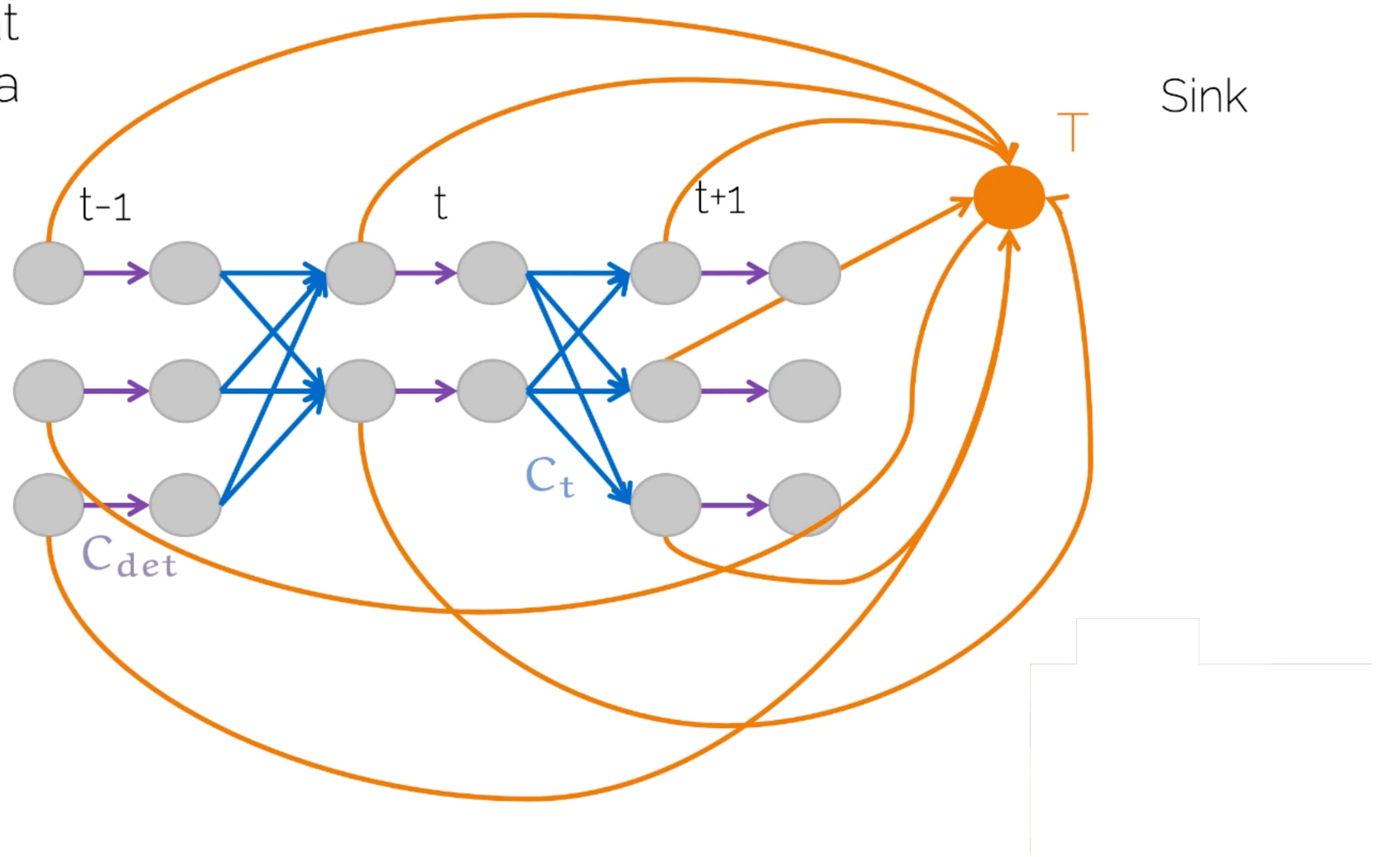
Connections that allow us to start a trajectory

https://www.youtube.com/watch?v=BR3Y5bAz5Dw&t

Tracking with network flows

Complete graph

Connections that allow us to end a trajectory



https://habr.com/ru/company/recognitor/blog/505694/

Super useful practical article from experienced engineer, but hard to read without a prior knowledge

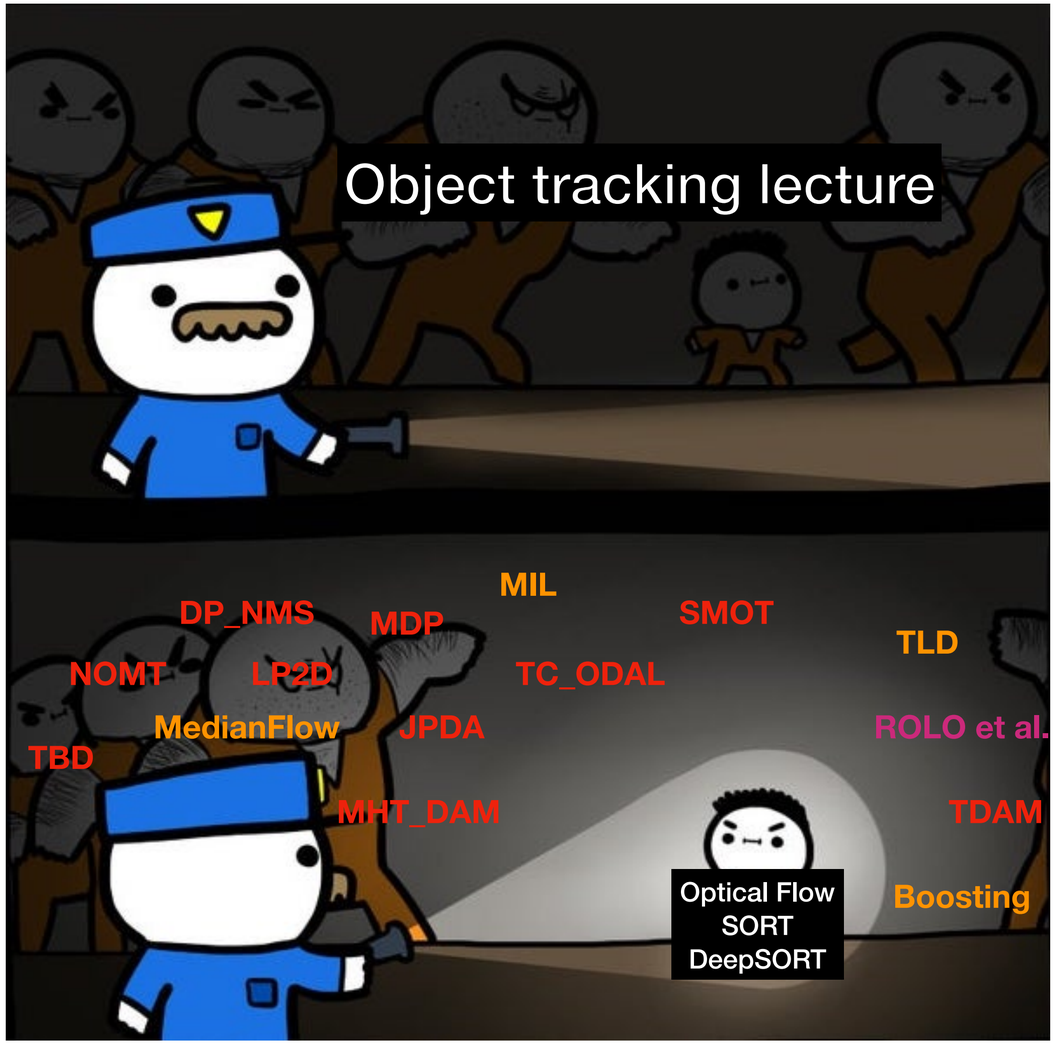
Your task

- Take any pre-trained Object Detector (my suggestion: YOLOv5 [my demo])
- Use any framework for tracking (my suggestions: SORT, DeepSORT, Norfair)
- Perform tracking of people on the standard OpenCV testing video
- Send your **code** + **video result** (*10 points*) me in Telegram and **present it** in 2 mins in Google Meet or Zoom (*10 points*)
- **Answer my questions** (*5 points*)

blank.py explanation

```python
def main(input_path, save_path, weights_path, device):
    start_time = time.time()
    model = Model(weights_path, device=device, classes=[0])

    dataset = LoadImages(input_path, img_size=640)

    fps = dataset.cap.get(cv2.CAP_PROP_FPS)
    w = int(dataset.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    h = int(dataset.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*"DIVX"), fps, (w, h))

    for frame_number, (path, img, im0s, vid_cap) in enumerate(dataset, 1):
        img_to_draw = im0s.copy()
        det = model.predict(img, im0s)
        print(f'detected {len(det[:, -1])} people')

        # ================================
        # todo: add your tracking somewhere here
        # ================================

        if det is not None:
            for *xyxy, conf, cls_id in det:
                # plotting bboxes of detected people
                plot_one_box(xyxy, img_to_draw, color=(0, 0, 255), line_thickness=2,
                             label=model.names[int(cls_id)])  # todo: add person id to label

        vid_writer.write(img_to_draw)

    vid_writer.release()

    print('Finished in {:.3f}s'.format(time.time() - start_time))
```

Summary


Object tracking lecture
MIL
DP_NMS
MDP
SMOT
TLD
NOMT
LP2D
TC_ODAL
MedianFlow
JPDA
ROLO et al.
TBD
MHT_DAM
TDAM
Optical Flow
SORT
DeepSORT
Boosting

Thanks for your attention