102b-object-tracker-HL-2020-4-16.odp

Example

Version: x0.1 (Alpha)
Date: Mar 6, 2020
Project Lead: Harry Li, Ph.D.

Team members:
Minh Duc Ong

ObjectID Tracker

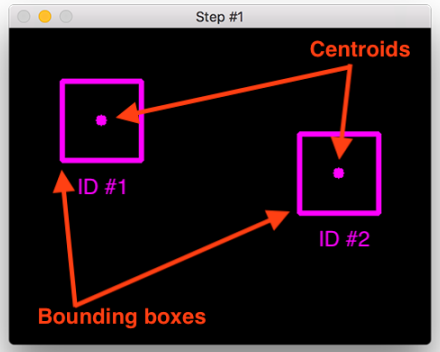
https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

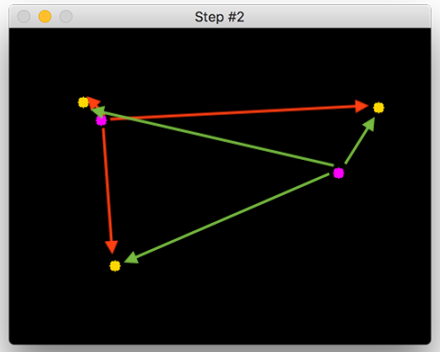
The Process of Object ID tracking is:

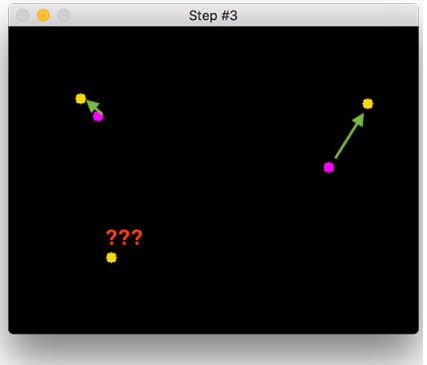
1. Taking an initial set of object detections (such as an input set of bounding box coordinates).
2. Creating a unique ID for each of the initial detections.
3. And then tracking each of the objects as they move around frames in a video, maintaining the assignment of unique IDs.

Algorithm:

Accept bounding box coordinates and compute centroids of each face object → Compute Euclidean distance between new bounding boxes and existing objects → Update (x, y)-coordinates of existing object → Register new objects → Deregister old objects





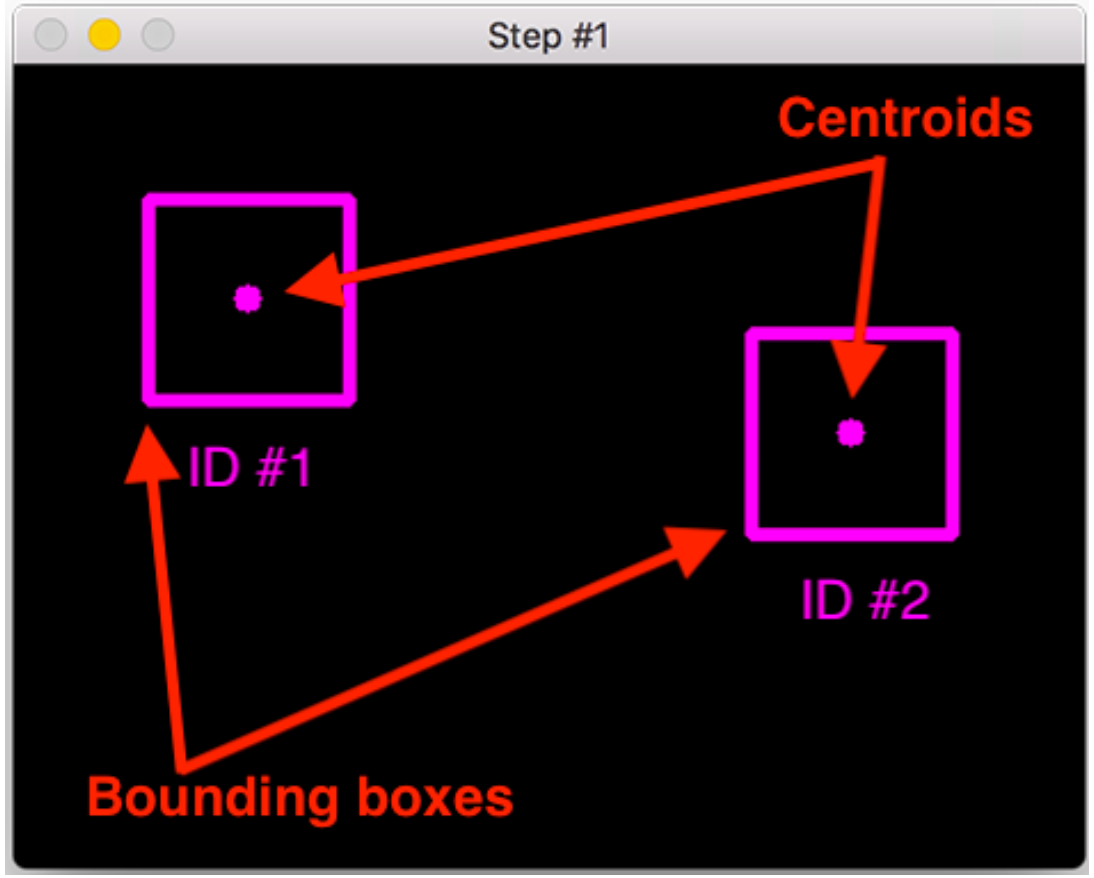


Step 0. Compute Bounding Box

These bounding boxes can be produced by object detector of your choice:
1. color thresholding + contour extraction;
2. Haar cascades;
3. HOG + Linear SVM; (HOG: Histogram of Oriented Gradient, https://www.learnopencv.com/tag/hog/)
4. SSDs; (single shot detection https://honingds.com/blog/ssd-single-shot-object-detection-mobilenet-opencv/)
5. Faster R-CNNs;
6. Yolo (you look only once).

They are computed for every frame in the video.

Step 1. Compute Centroid



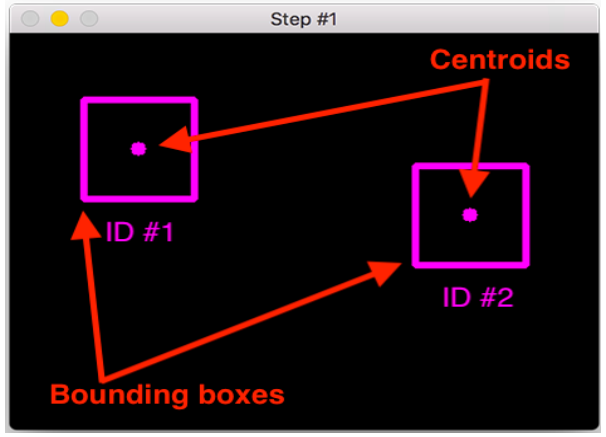
The algorithm:

1. Compute centroids, computation is given in my lecture on binary image processing and moments computation

2. Assign ID to each centroid (box)

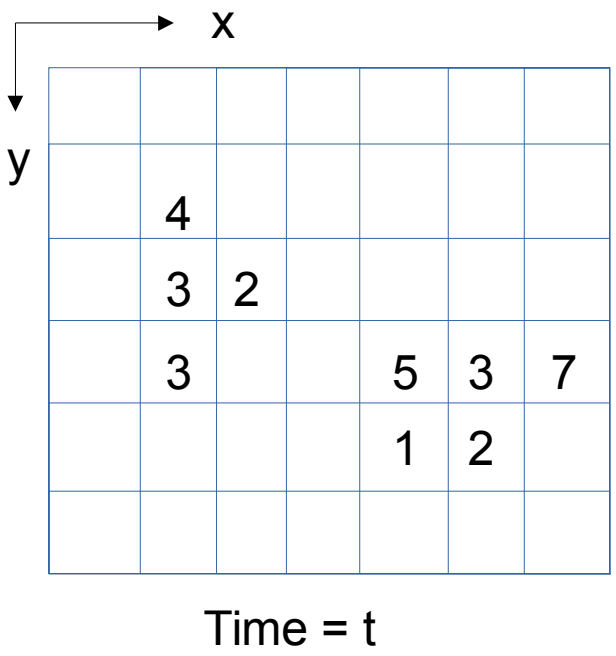
Centroid computation reference: 2019S-24-2018S-114-Contour-Inference-final-2018-4-30.pdf

Example Calculation Part 1

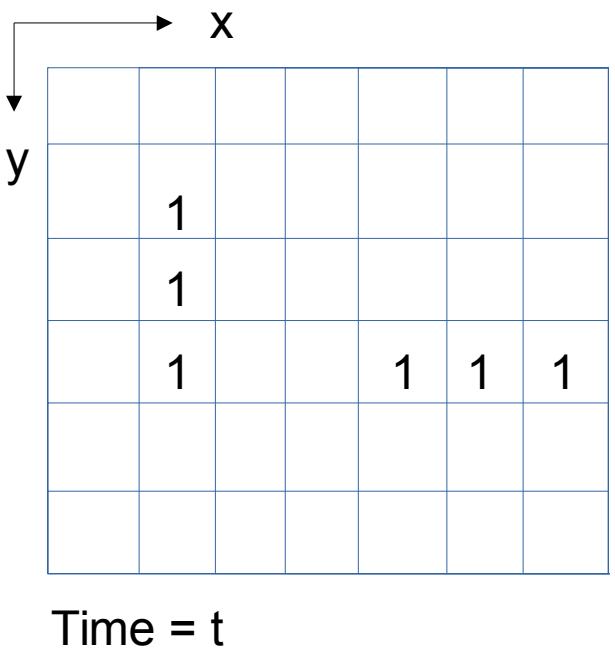


Example:
Step 0: detection of the object using intensity thresholding technique, set Threshold T = 3, for I(x,y) >= T, set it to 1, o/w to 0.
Step 1. Compute centroids,

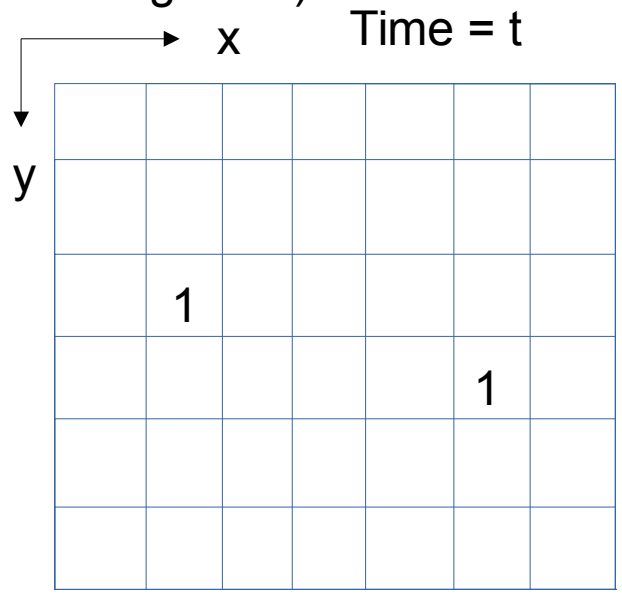
x1_bar = 1, y1_bar = 2. Assign ID as O1, similarly, x2_bar = 5, y2_bar = 3. Assign ID as O2. Then create (update the tracking table)



Time = t



Time = t



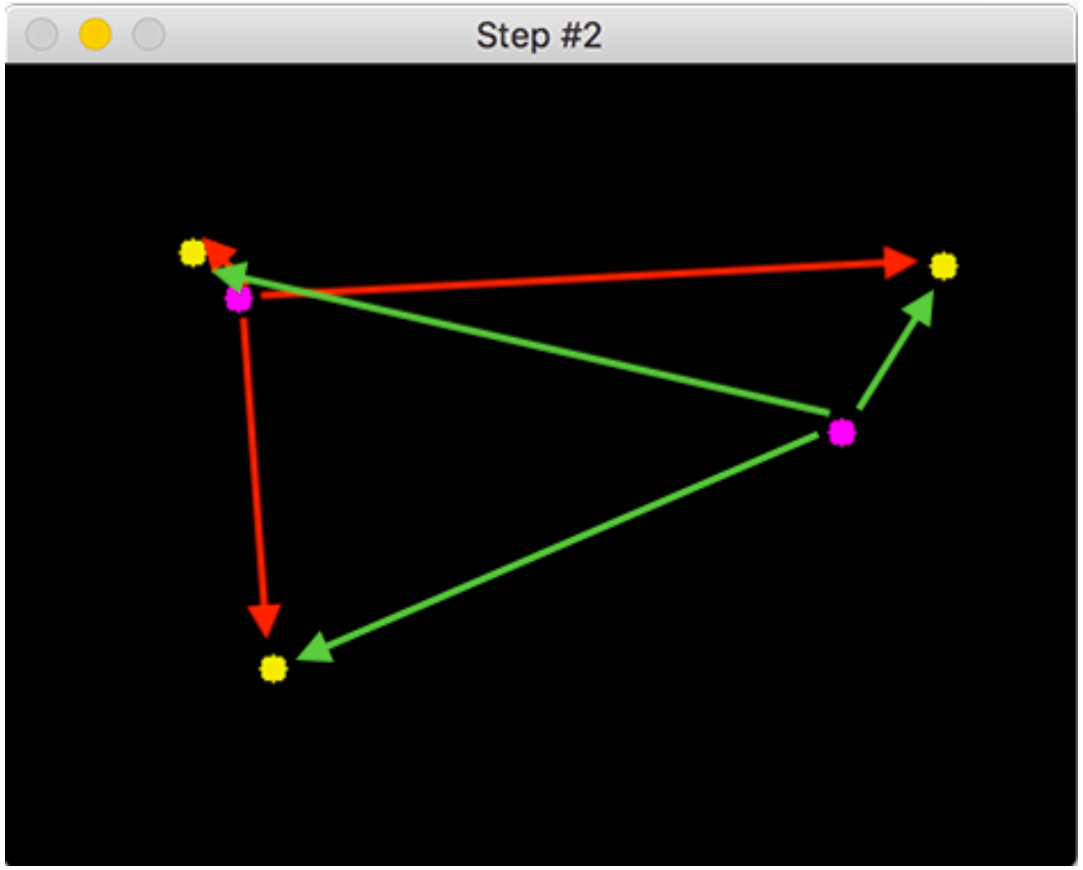
Time = t

Registration table

Obj. No.	ID	x-bar, y-bar
Object 1.	1	1, 2
Object 2.	2	5, 3

Step 2. Compute Distance Between Boxes

O1



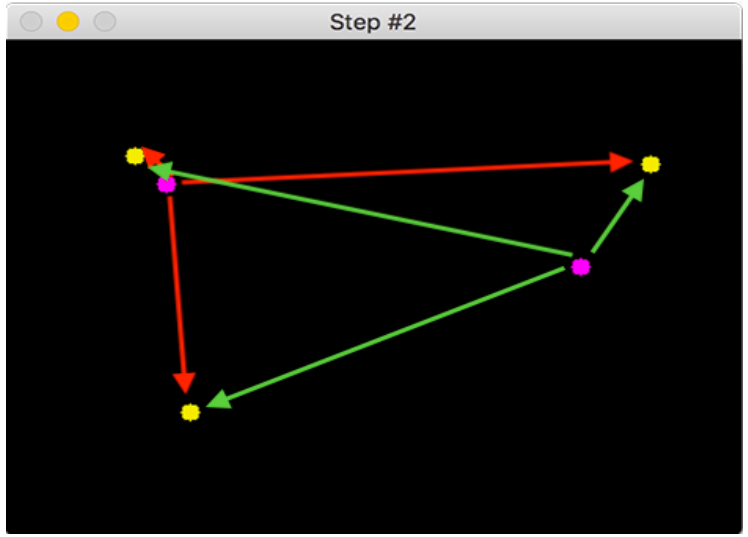
O2

Original: pink (from previous frame)

New: yellow (the current frame)

Algorithm: compute distance between each pair

Example Part 2



Example:
Step 2: detection of the object at time t+dt using the same technique, binarized image I(x,y). Compute the centroids, as x1_bar = 0, y1_bar = 1, x2_bar = 6, y2_bar = 2, x3_bar = 1, y3_bar = 4. Compute distance with O1, O2 as reference points.

1						1
	1					
					1	
	1					

D(o1, o1_new) = sqrt(2); D(o1, o2_new) = sqrt(26); D(o1, o3_new) = sqrt(4);
D(o2, o1_new) = sqrt(29); D(o2, o2_new) = sqrt(5); D(o1, o3_new) = sqrt(17);

x

y

3						9
4	2				2	3
3					1	3
						2
2	4	2				
	1					

Time = t+dt

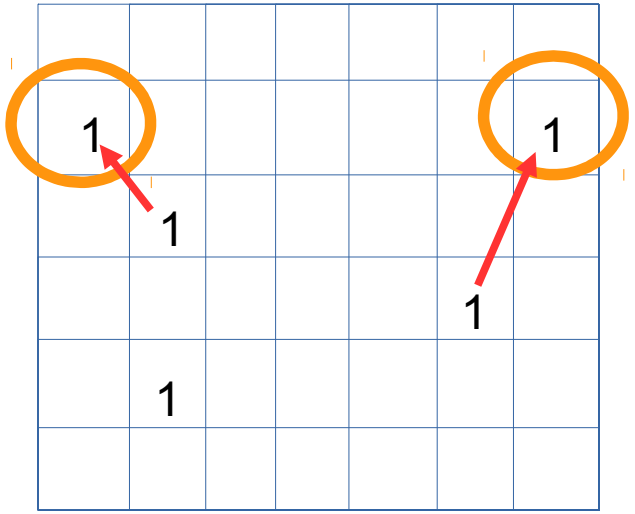
1						1
1						1
1						1
	1					

Time = t+dt

Temporary Registration table time = t+dt

Obj. No.	ID	x-bar, y-bar
Object 1.	1	0, 1
Object 2.	2	6, 1
Object 3.	3	1, 4

Step 2. Calculation



Minimum distance from o1:

Min D = sqrt(2), D(o1, o1_new) = sqrt(2), so the matching is o1_new (in Yellow circle)

Minimum distance from o2:

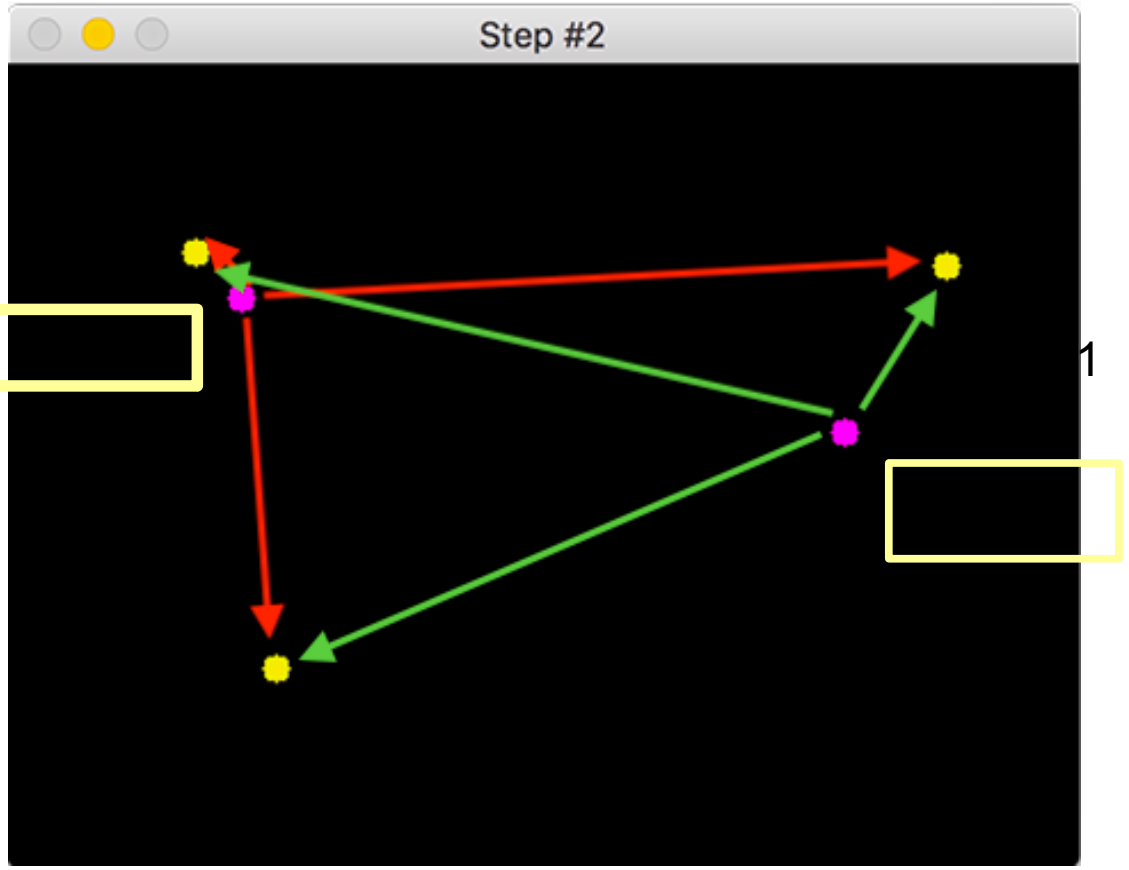
Min D = sqrt(5), D(o2, o2_new) = sqrt(5); so the matching is o2_new (in Yellow circle)

Update the registration table

Registration table (time = t + dt)

Obj. No.	ID	x-bar, y-bar
Object 1.	1	0, 0
Object 2.	2	6, 1
Object 3.	3	1, 4

Step 3. Compute Distance Between Boxes

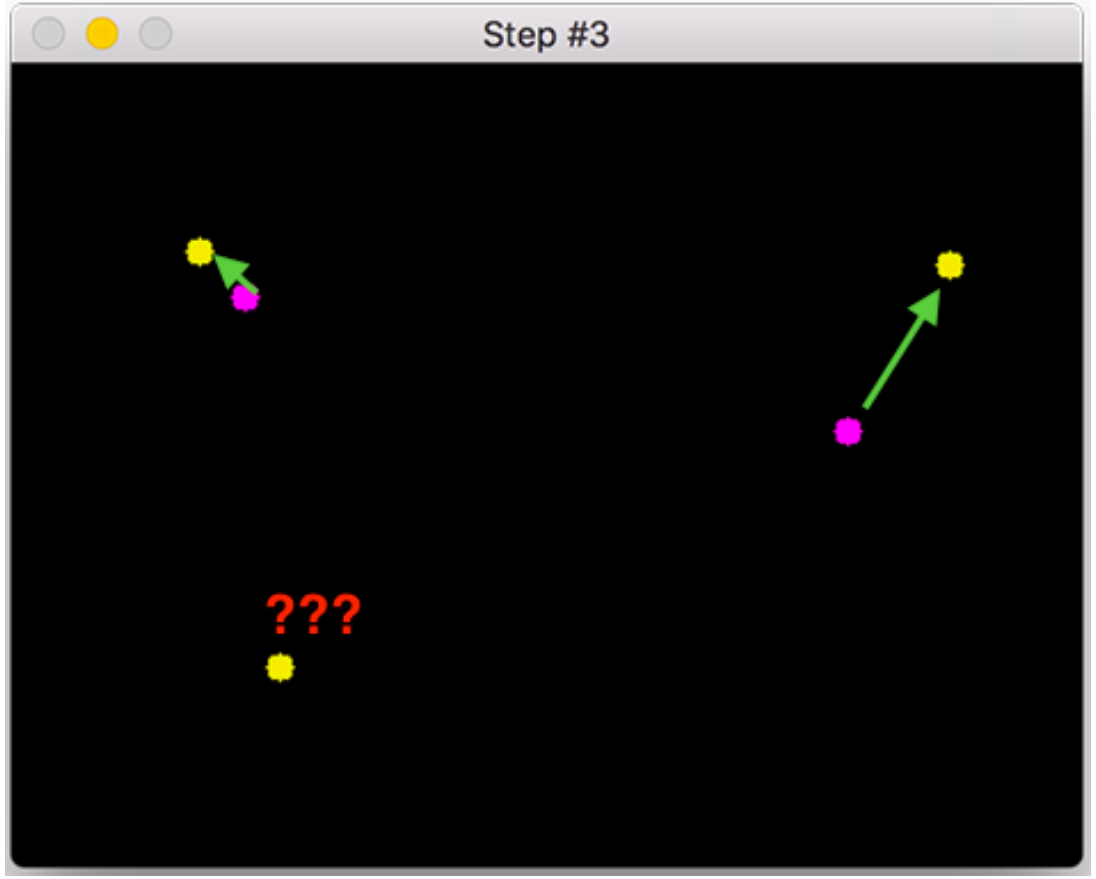


Original: pink (from previous frame)

New: yellow (the current frame)

Algorithm: compute distance between each pair

Step 4. Update Box ID with Shortest Distance



Original: pink (from previous frame)

New: yellow (the current frame)

Algorithm: find shortest distance to match the pair.

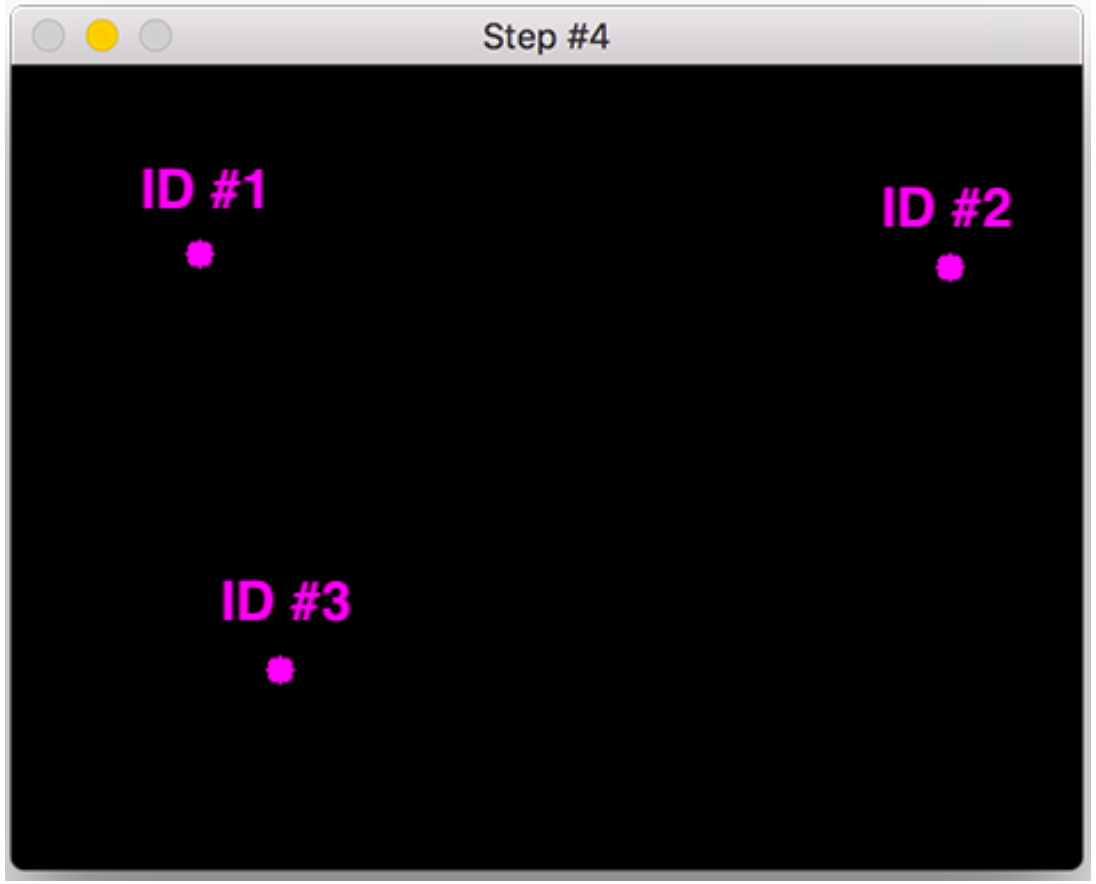
Example:

O1 > N1

O2 > N2

And New N3 appeared on this frame

Step 5. Add New Object ID then Back to Step 2



Algorithm: add ID to the new object, e.g., register the new object ID.

Example:

Object 1, with ID O1
Object 2, with ID O2

And New object ID: O3

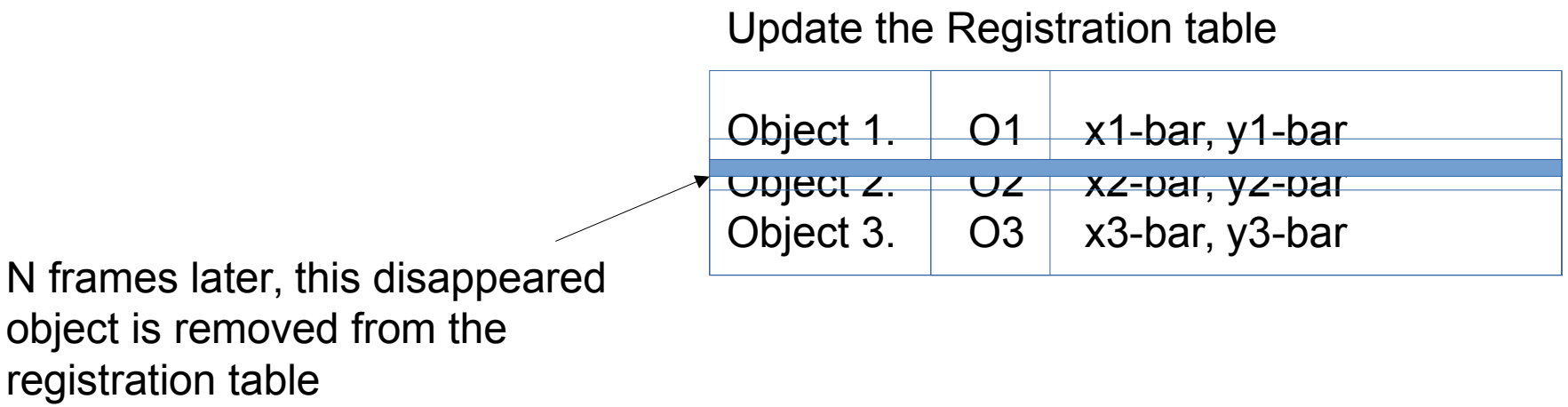
Registration table

Object 1.	O1	x1-bar, y1-bar
Object 2.	O2	x2-bar, y2-bar
Object 3.	O3	x3-bar, y3-bar

Step 6. Remove Object ID in N Frames

Object tracking needs to be able to handle when an object has been (1) lost, (2) disappeared, or (3) left the field of view.

e.g. will deregister old objects when they cannot be matched to any existing objects for a total of N subsequent frames.



Sample Code

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

Structure of the distribution files

Simple object tracking with OpenCV

```
$ tree --dirsfirst
.
├── pyimagesearch
│   ├── __init__.py
│   └── centroidtracker.py
├── object_tracker.py
├── deploy.prototxt
└── res10_300x300_ssd_iter_140000.caffemodel

1 directory, 5 files
```

You can replace the centroid tracker based on the techniques in ROI localization

The remaining .prototxt and .caffemodel files are part of the OpenCV deep learning face detector. They are necessary for today's face detection + tracking method, but you could easily use another form of detection (more on that later).

ObjectID Tracker Algorithm

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

An ideal object tracking algorithm will:
1. Only require the object detection phase once (i.e., when the object is initially detected)
2. Will be extremely fast — much faster than running the actual object detector itself
3. Be able to handle when the tracked object “disappears” or moves outside the boundaries of the video frame
4. Be robust to occlusion
5. Be able to pick up objects it has “lost” in between frames