This implementation is about designing a robust *multi*-*object tracking* (MOT) framework. The general architecture of this work include three main subsections:

1. object detections (creating sets of bounding box coordinates for each detected object in each input frame)
2. Creating a unique ID for each of the initial detections
3. tracking module, which tracks each of the objects as they move around frames in a video, maintaining the assignment of unique IDs

Furthermore, object tracking allows us to **apply a unique ID to each tracked object**, making it possible to count individual objects in a video.

To build a real-time object tracking algorithm, we should consider the algorithm's speed improvement besides maintaining its accuracy. As a result, our object tracking module will:

1. Only require the object detection at the first frame that it appears (i.e., when the object is initially detected)
2. Be able to handle when the tracked object "disappears" or moves outside the boundaries of the video frame.
3. Be robust to occlusion.
4. Be able to pick up objects it has "lost" in-between frames.

We implement centroid tracking with OpenCV as the fundamental of our object tracking methodology.

It is easy to understand yet highly effective tracking algorithm, one of the core kernel-based and correlation-based tracking algorithms.

# *1.object detection*

This module provides the feed of the centroid tracking modules.

we detect moving objects in each video frames and drawing a bounding box or region of the interest (ROI) box around them,

The following models were used as object trackers in this study:

SSDMobileNet

SSDMobilenetV2

YOLOv3

HOG+SVM

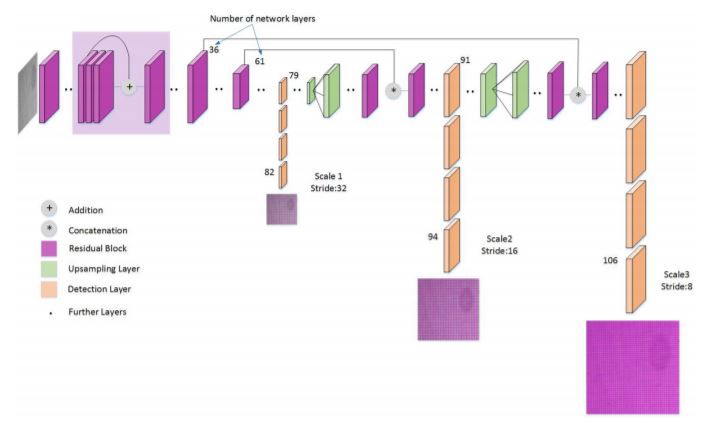
The best module in terms of accuracy was YOLOv3 (Joseph Redmon et al. 2018), as was expected.

The “You Only Look Once,” or Yolov3, as described in its original paper. family of models is a series of end-to-end deep learning models designed for fast object detection, developed by Joseph Redmon et al. and first described in the 2015 paper titled “You Only Look Once: Unified, Real-Time Object Detection.”

The approach involves a single deep convolutional neural network (originally a version of GoogLeNet, later updated and called DarkNet based on VGG) that splits the input into a grid of cells. Each cell directly predicts a bounding box and object classification. A result is a large number of candidate bounding boxes that are consolidated into a final prediction by a post-processing step.

There are four main variations of the approach; they are YOLOv1, YOLOv2, YOLOv3, and YOLOv4. The first version proposed the general architecture, whereas the second version refined the design and made use of predefined anchor boxes to improve the bounding box proposal. Version three further refined the model architecture and training process.

They are famous for object detection because of their detection speed, often demonstrated in real-time on video or with camera feed input.

**Figure 1:** the first step is to accept bounding box coordinates from an object detector and use them to compute centroids.

# *2.centroid tracking*

This object tracking algorithm relies on the Euclidean distance between (1) *existing* object centroids (i.e., objects the centroid tracker has already seen before) and (2) new object centroids between subsequent frames in a video.

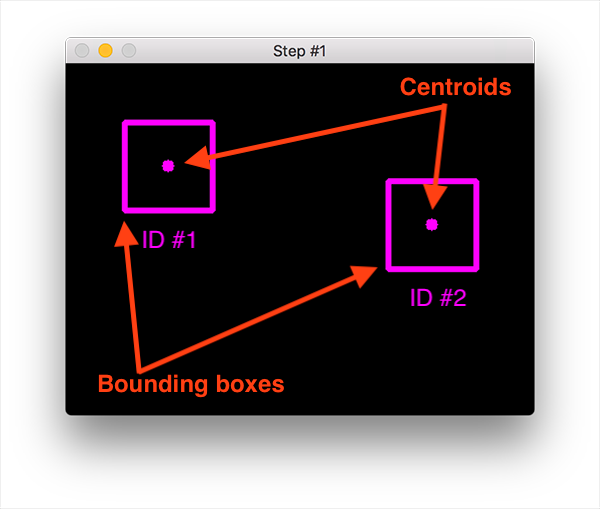
We'll review the centroid algorithm in more depth in the following section. From there, we'll implement a Python class to contain our centroid tracking algorithm and then create a Python script to run the object tracker and apply it to input videos.

Finally, we'll run our object tracker and examine the results, noting both the positives and the algorithm's drawbacks.

### The centroid tracking algorithm

The centroid tracking algorithm is a multi-step process. We will review each of the tracking steps in this section.

## Step #1: Accept bounding box coordinates and compute centroids

**Figure 1:** the first step is to accept bounding box coordinates from an object detector and use them to compute centroids.

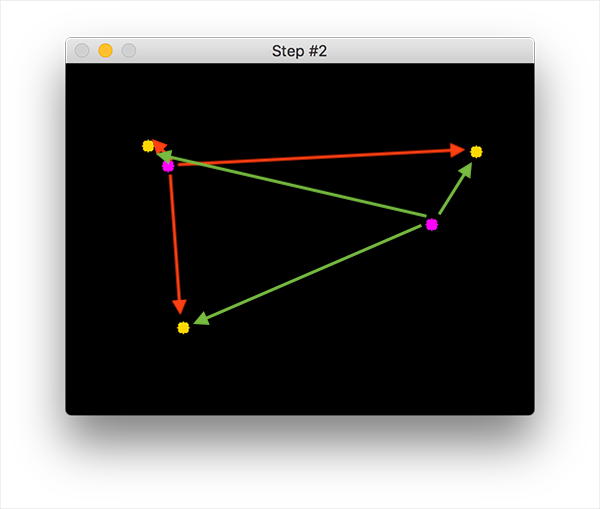
The centroid tracking algorithm assumes that we pass in a set of the bounding box *(x, y)*-coordinates for each detected object in ***every single frame***.

Any type of object detector can produce these bounding boxes (e.g., color thresholding + contour extraction, Haar cascades, HOG + Linear SVM, SSDs, Faster R-CNNs, etc.), provided that they computed for every frame in the video.

Once we have the bounding box coordinates, we must compute the "centroid," or more simply, *the center (x, y)-coordinates* of the bounding box. **Figure 1** above demonstrates accepting a set of bounding box coordinates and computing the centroid.

Since these are the first initial set of bounding boxes presented to our algorithm, we will assign them unique IDs.

## Step #2: Compute Euclidean distance between new bounding boxes and existing objects



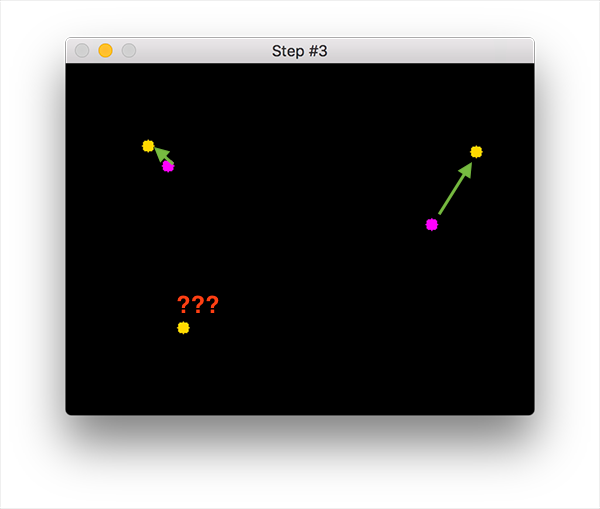
**Figure 2:** Three objects are present in this image for simple object tracking with Python and OpenCV. We need to compute the Euclidean distances between each pair of original centroids (*red*) and new centroids (*green*).

For every frame in our video stream, we apply **Step #1** of computing object centroids; however, instead of assigning a new unique ID to each detected object, we first need to determine if we can *associate* the *new* object centroids (yellow) with the *old* object centroids (purple). To accomplish this process, we compute the Euclidean distance (highlighted with green arrows) between each pair of existing object centroids and input object centroids.

From **Figure 2**, you can see that we have this time detected three objects in our image. The two pairs that are close together are two existing objects.

We then compute the Euclidean distances between each pair of original centroids (yellow) and new centroids (purple). But how do we use the Euclidean distances between these points to match them and associate them?

The answer is in **Step #3**.



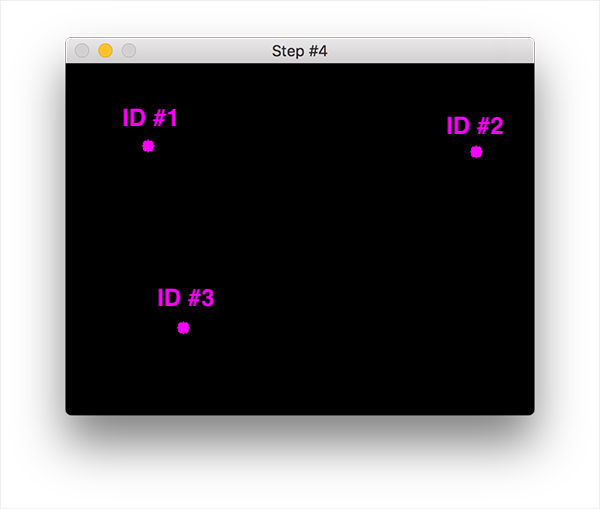
**Figure 3:** Our simple centroid object tracking method has associated objects with minimized object distances.

The centroid tracking algorithm's primary assumption is that a given object will potentially move in between subsequent frames. Still, the *distance* between the centroids for frames Ft+1 and Ft will be smaller than all other distances between objects.

Therefore, if we choose to associate centroids with minimum distances between subsequent frames, we can build our object tracker.

In **Figure 3**, our centroid tracker algorithm chooses to associate centroids that minimize their respective Euclidean distances.

## Step #4: Register new objects



**Figure 4:** In our object tracking with Python and OpenCV example, we have a new object that wasn't matched with an existing object, so it is registered as object ID #3.

In the event that there are more input detections than existing objects being tracked, we need to register the new object. "Registering" means that we are adding the new object to our list of tracked objects by:

1. Assigning it a new object ID
2. Storing the centroid of the bounding box coordinates for that object

We can then go back to **Step 2** and repeat the pipeline of steps for every frame in our video stream.

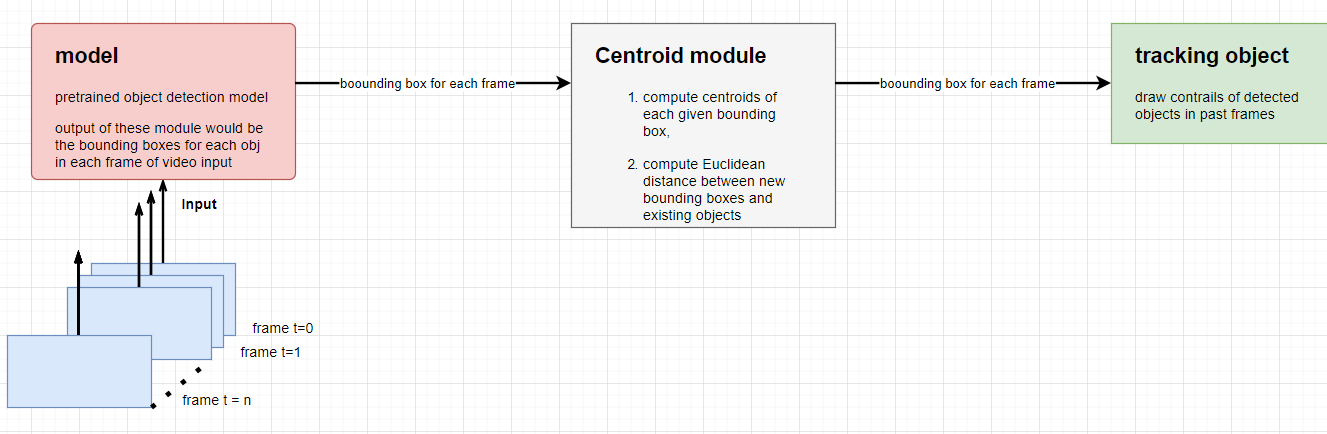
Figure 4 demonstrates the process of using the minimum Euclidean distances to associate existing object IDs and then registering a new object.

## Step #5: Deregister old objects

Any reasonable object tracking algorithm needs to be able to handle when an object has been lost, disappeared, or left the field of view.

Exactly how you handle these situations is really dependent on where your object tracker is meant to be deployed, but for this implementation, we will deregister old objects when they cannot be matched to any existing objects for a total of *N* subsequent frames.

# The overall architecture



**Figure 5**: In our object tracking with Python and OpenCV example, we have a new object that wasn't matched with an existing object, so it is registered as object ID #3.