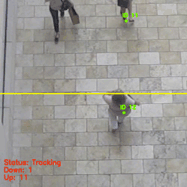
**PEOPLE COUNTER**

The main aim of this project is to build an application that can detect people in a live video feed and track their movement.

This can be used for a variety of purposes. The main application can be to count the number of people entering and exiting a facility. Currently the entire world is fighting against the coronavirus and to promote social distancing we have seen that only few people are allowed to be inside a store at a time, even at events like weddings only a certain number of people are allowed to attend. It is not always possible to manually keep a count of the people inside hence we can use this application to easily track the movement of people.

Although many devices are available in the market for this purpose like infrared counters, but these are very expensive. Our solution is a very cost effective one as the only hardware requirements are a CCTV camera which is already present in most such places.



# Understanding object detection vs. object tracking

When we apply object detection we are determining where in an image/frame an object is. An object detector is also typically more computationally expensive, and therefore slower, than an object tracking algorithm. Examples of object detection algorithms include Haar cascades, HOG + Linear SVM, and deep learning-based object detectors such as Faster R-CNNs, YOLO, and Single Shot Detectors (SSDs).

An object tracker, on the other hand, will accept the input (x, y)-coordinates of where an object is in an image and will:

Assign a unique ID to that particular object

Track the object as it moves around a video stream, predicting the new object location in the next frame based on various attributes of the frame (gradient, optical flow, etc.)

Examples of object tracking algorithms include MedianFlow, MOSSE, GOTURN, kernalized correlation filters, and discriminative correlation filters, to name a few.

# Combining both object detection and object tracking

Highly accurate object trackers will combine the concept of object detection and object tracking into a single algorithm, typically divided into two phases:

Phase 1 — Detecting: During the detection phase we are running our computationally more expensive object tracker to (1) detect if new objects have entered our view, and (2) see if we can find objects that were “lost” during the tracking phase. For each detected object we create or update an object tracker with the new bounding box coordinates. Since our object detector is more computationally expensive we only run this phase once every N frames.

Phase 2 — Tracking: When we are not in the “detecting” phase we are in the “tracking” phase. For each of our detected objects, we create an object tracker to track the object as it moves around the frame. Our object tracker should be faster and more efficient than the object detector. We’ll continue tracking until we’ve reached the N-th frame and then re-run our object detector. The entire process then repeats.

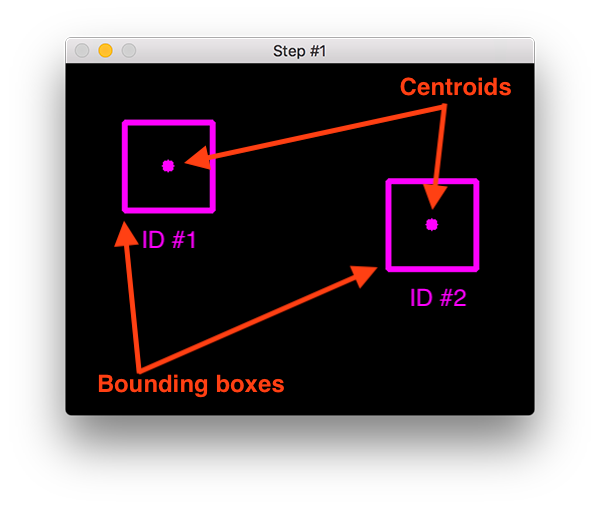
The benefit of this hybrid approach is that we can apply highly accurate object detection methods without as much of the computational burden. We will be implementing such a tracking system to build our people counter.

# Combining object tracking algorithms

To implement our people counter we’ll be using both OpenCV and dlib. We’ll use OpenCV for standard computer vision/image processing functions, along with the deep learning object detector for people counting.

We’ll then use dlib for its implementation of correlation filters.

At Step #1 we accept a set of bounding boxes and compute their corresponding centroids (i.e., the center of the bounding boxes):

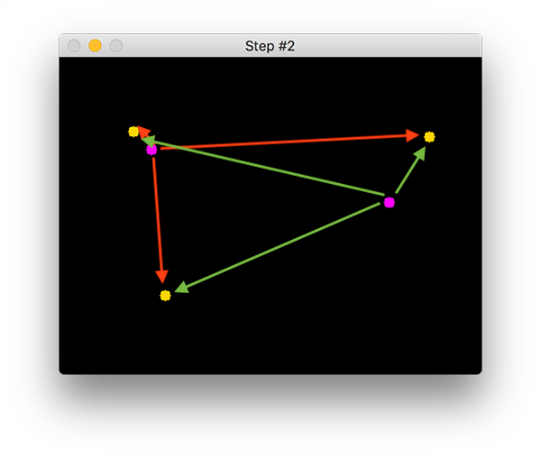


The bounding boxes themselves can be provided by either:

An object detector (such as HOG + Linear SVM, Faster R- CNN, SSDs, etc.)

Or an object tracker (such as correlation filters)

During Step #2 we compute the Euclidean distance between any new centroids (yellow) and existing centroids (purple):

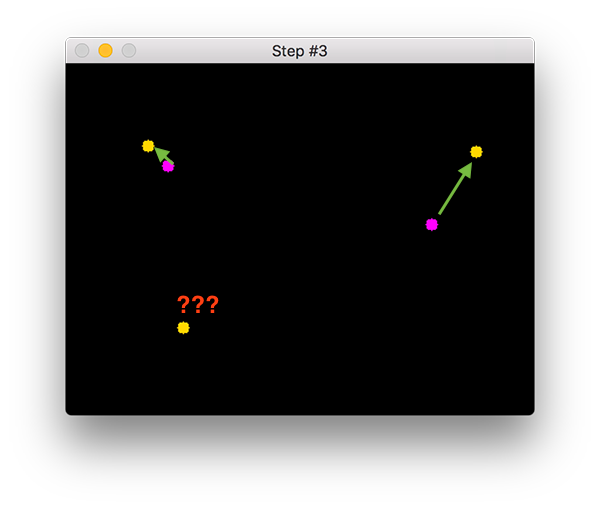


The centroid tracking algorithm makes the assumption that pairs of centroids with minimum Euclidean distance between them must be the same object ID.

In the example image above we have two existing centroids (purple) and three new centroids (yellow), implying that a new object has been detected (since there is one more new centroid vs. old centroid).

The arrows then represent computing the Euclidean distances between all purple centroids and all yellow centroids.

Once we have the Euclidean distances we attempt to associate object IDs in Step #3:

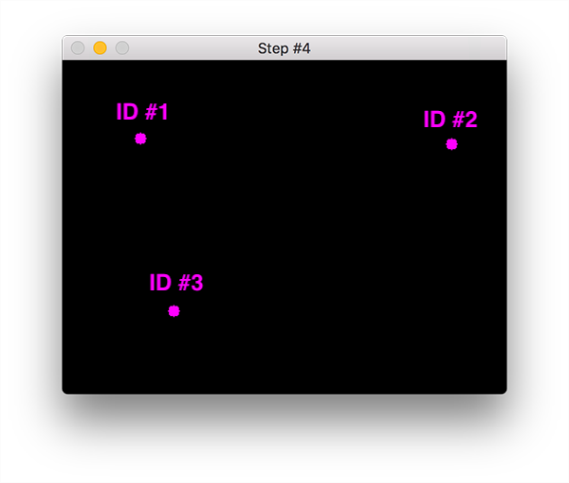


In the Figure above you can see that our centroid tracker has chosen to associate centroids that minimize their respective Euclidean distances.

But what about the point in the bottom-left?

It didn’t get associated with anything — what do we do?

To answer that question, we need to perform Step #4, registering new objects:



Registering simply means that we are adding the new object to our list of tracked objects by:

Assigning it a new object ID

Storing the centroid of the bounding box coordinates for the new object

In the event that an object has been lost or has left the field of view, we can simply deregister the object (Step #5).

Exactly how you handle when an object is “lost” or is “no longer visible” really depends on your exact application, but for our people counter, we will deregister people IDs when they cannot be matched to any existing person objects for 40 consecutive frames.

# Creating a “trackable object”

In order to track and count an object in a video stream, we need an easy way to store information regarding the object itself, including:

It’s object ID

It’s previous centroids (so we can easily to compute the direction the object is moving)

Whether or not the object has already been counted

To accomplish all of these goals we can define an instance of TrackableObject — open up the trackableobject.py file and insert the following code:

OpenCV People Counter

class TrackableObject:

def \_\_init\_\_(self, objectID, centroid):

# store the object ID, then initialize a list of centroids

# using the current centroid

self.objectID = objectID

self.centroids = [centroid]

# initialize a boolean used to indicate if the object has

# already been counted or not

self.counted = False

The TrackableObject constructor accepts an objectID + centroid and stores them. The centroids variable is a list because it will contain an object’s centroid location history.

The constructor also initializes counted as False , indicating that the object has not been counted yet.

For further reference: <https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/>