Congestion Models and APIs

Project 1: Congestion model using Computer Vision

Objective : To implement people counters using both OpenCV and dlib. OpenCV for standard computer vision/image processing functions, along with the deep learning object detector for people counting. then use dlib for its implementation of correlation filters.

Requirements:- In order to build our people counting applications, we'll need a number of different Python libraries, including:

- NumPy
- OpenCV
- Dlib
- Imutils

Abstract:-

"People_Counter" with OpenCV and Python. Using OpenCV, we'll count the number of people who are heading "in" or "out" of a bus or metro or store department store in real-time. Using an SSD model to detect the person label class and then track using an algorithm is cenroids_tracker.

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:

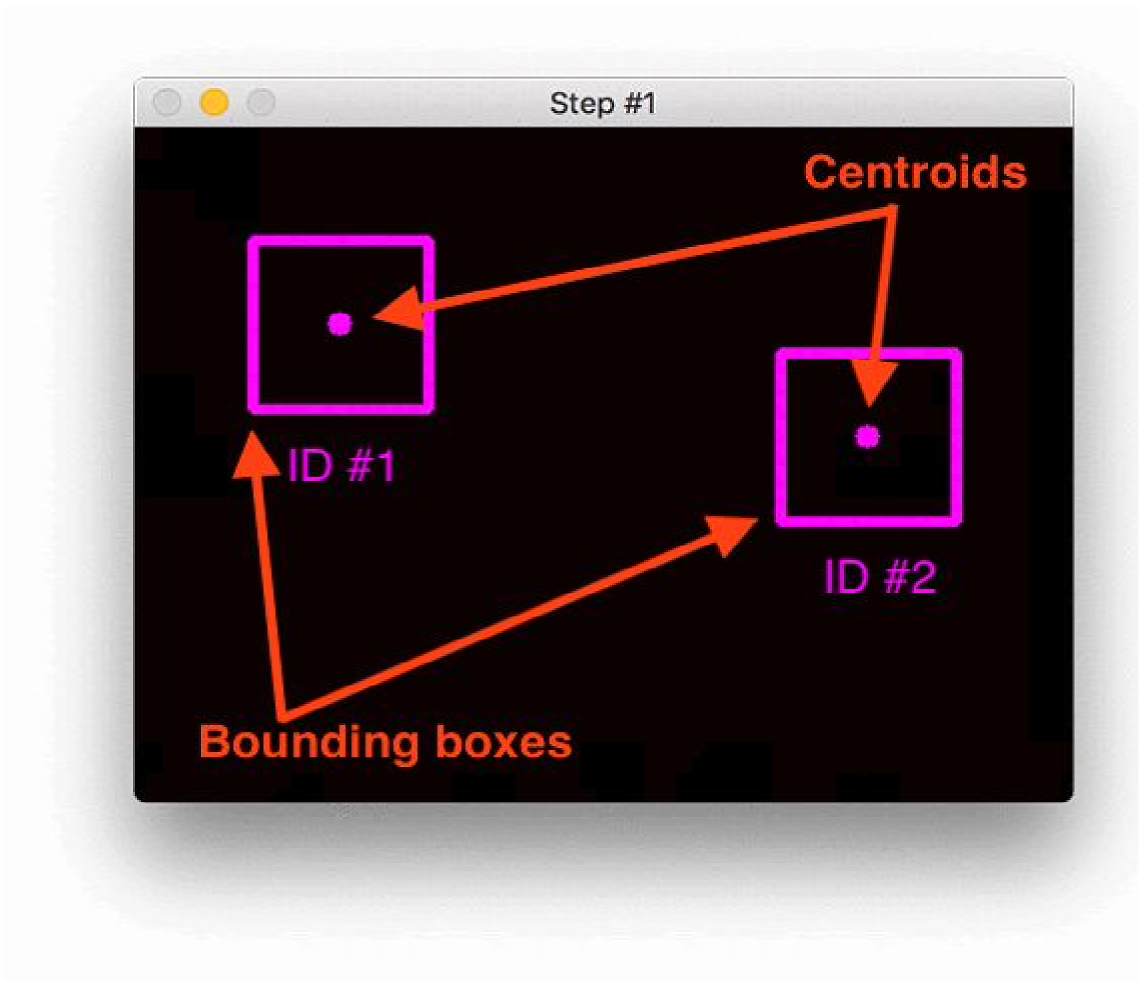
1. Assign a unique ID to that particular object
2. Track the object as it moves around a video stream, *predicting* the new object location in the next frame based on various attributes of the frame (gradient, optical flow, etc.)

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:

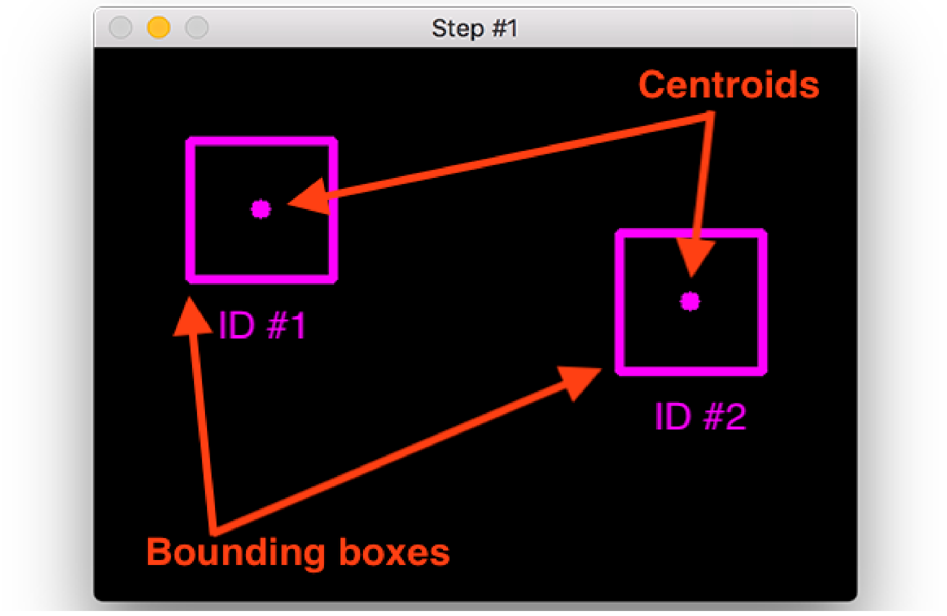
Phase 1 — Detecting: During the detection phase we are running our computationally more expensive object tracker to (1) detect if new objects have entered our view, and (2)see if we can find objects that were "lost" during the tracking phase. For each detected object we create or update an object tracker 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.



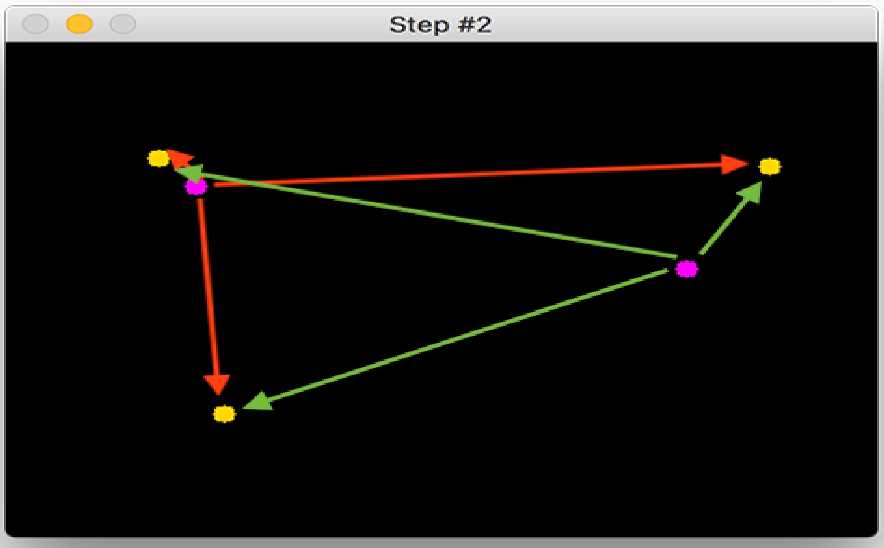
Centroid Tracking Algorithm

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

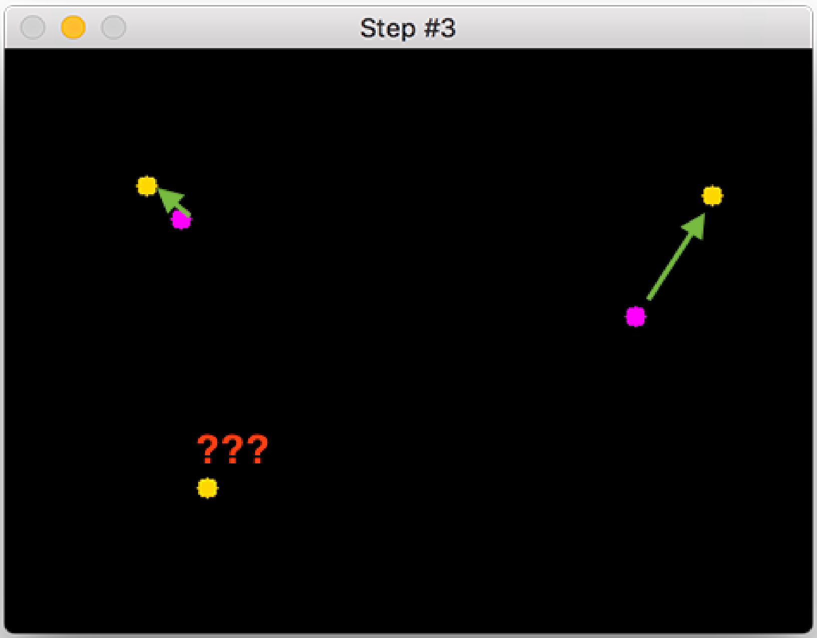


- **Step 2:** Compute the Euclidean distance between any new centroids (yellow) and existing centroids (purple):

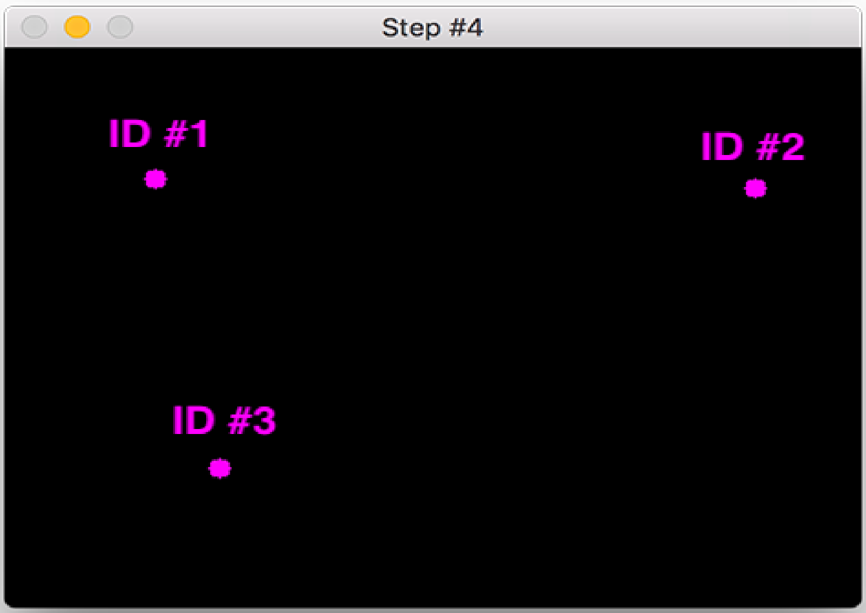
Three objects are present in this image. We need to compute the Euclidean distance between each pair of original centroids (red) and new centroids (green).



-
- **Step 3:** Once we have the Euclidean distances we attempt to associate object IDs



- **Step 4:** registering new objects:



- **Step 5**: an object has been lost or has left the field of view, we can simply deregister the object
- Registering simply 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 the new object.

Github: https://github.com/rehanfazalkhan/People_Counter_SSD