Object Tacking

Aniket Bhatia
aniketb@iitb.ac.in

Vineet Kotariya
16D110009@iitb.ac.in

1 Introduction

Tracking an object (or even multiple objects for that matter) has become an important task. We use tracking in surveillance cameras, traffic density estimation and even in vehicles for collision avoidance. This motivates the project in this domain. Tracking can be achieved using conventional approaches and also by the recent advent in deep learning. We first explore tracking using non-deep methods and then move on to deep learning methods. Deeo learning methods have provided us with tremndous gains in accuracy and performance of the detection and tracking approaches. Most of the state-of-the art approaches follow the 'tracking by detection' technique where in they first find the object in the first frame and find its position in the next frame.

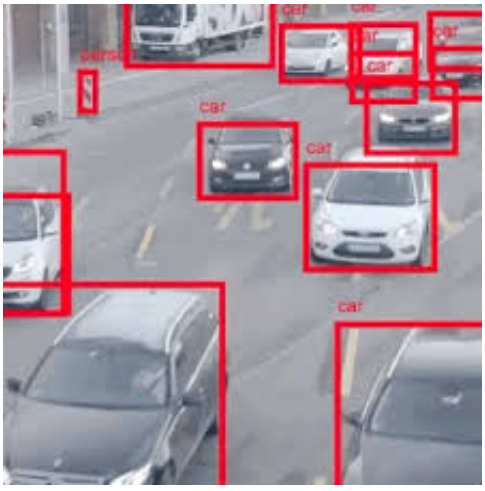

Figure 1: Tracking cars in traffic, source: here

2 Challenges

When one approaches a tracking problem many challenges come in the way. These issues include variations due to geometric changes (eg: Pose, scale of objects, deformation in the subsequent frames), difference due to photometric factors (eg: illumination, appearance), non-linear motion, limited resolution, similarity of objects in the scenes, very crowded settings, tracking initiation and termination, objects crossing paths leading to object IDs getting switched, etc.

3 Approaches

3.1 Mean Shift Tracking

The intuition behind the meanshift is simple. Consider you have a set of points. (It can be a pixel distribution like histogram backprojection). You are given a small window (may be a circle) and you have to move that window to the area of maximum pixel density (or maximum number of points). An initial window is chosen around an initial choice of center. But if we find the centroid of the points inside that window, we will get a point, which is the real centroid of the window. Surely the original center, around which the window was chosen, and the new center don't match. So we move our window such that the circle of the new window matches with the previous centroid. Again, we find the new centroid. Most probably, it won't match. So we move it again, and continue the iterations such that the center of window and its centroid falls on the same location (or within a small desired error). So finally what we obtain is a window with maximum pixel distribution. So here for tracking we pass the histogram backprojected image and initial target location. When the object moves, obviously the movement is reflected in the histogram backprojected image. As a result, the meanshift algorithm moves our window to the new location with maximum density, thereby achieving tracking. The mean-shift algorithm is an efficient approach to tracking objects whose appearance is defined by histograms.

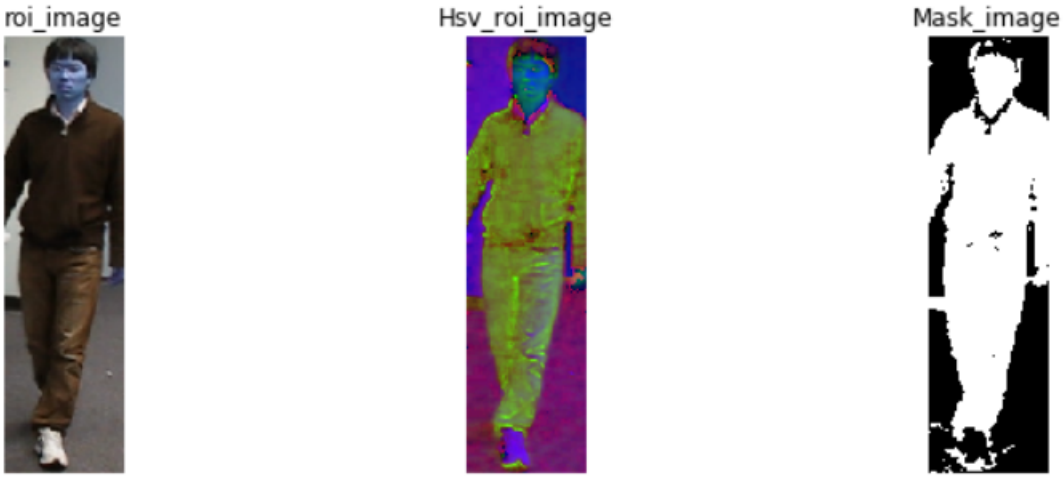

Figure 2: The ROI (Region of interest) in HSV and its mask

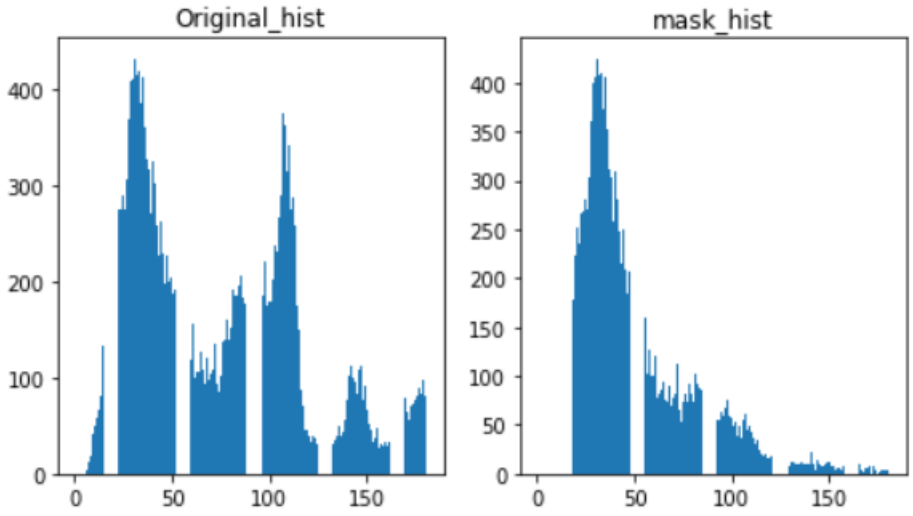

Figure 3: Histograms for the original image and the mask

Figure 4: HSV for the first frame

Figure 5: The backprojected image

3.2 Using Optical Flow

The actual or observed(relative motion) between objects and observer(camera) is known as optical flow. If the camera is moving and the object is stationary then also we will stay that we have optical flow. Optical flow methods are used to estimate motion of objects between two consecutive image frames. For example, let an object when seeming to move say, left. For our system to be able to understand that the object is moving to the left, it would be useful to find a way to add vectors to the can (known as flow vectors) which point to the left, thus describing its motion. Given two consecutive frames, how can we find the flow vectors for the first frame which describe how objects move between frames? To start, we make a reasonable assumption called the brightness constancy assumption: the pixel intensity of a moving point stays the same between two consecutive frames with small time difference. In other words, picking any pixel of the moving can, its brightness stays approximately the same between frames–its movement should not affect its brightness after all. Tracking using optical flow is made possible by the Lucas Kanade algorithm. One issue with the optical flow equation is that there are two unknowns that we want to solve for. This problem is known as the aperture problem. In other words, just looking an "aperture" at one pixel at a time, it is impossible to discern the true direction of motion of the object in question. The Lucas–Kanade method solves this problem by adding another assumption: spatial coherence. That is, that the motion of the image contents between two frames is approximately constant within a neighborhood of the point under consideration.

Now for tracking using the Lucas Kanade algorithm we first need to find keypoints to track. Harris corner detector is commonly used to initialize the keypoints to track with Lucas-Kanade method. Now we can use Lucas-Kanade method to track keypoints across multiple frames. The idea is simple: compute flow vectors at keypoints in the i^{th} frame, and add the flow vectors to the points to keep track of the points in $i+1^{th}$ frame. We notice that some of the points just drift away and are not tracked very well. Instead of keeping these 'bad' tracks, we would want to somehow declare some points are 'lost' and just discard them. One simple way to is to compare the patches around tracked points in two subsequent frames. If the patch around a point is not similar to the patch around the corresponding point in the next frame, then we declare the point to be lost. Here, we used the mean squared error between two normalized patches as the criterion for lost tracks.

Figure 6: Keypoints detected in the first frame

Figure 7: Optical flow vectors from the first frame to the second

We did not obtain that good results with the conventional Lucas Kanade an so we decided to use an iterative approach for the same. The optical flow vectors for the same are as shown below.

Figure 8: Optical flow vectors from the first frame to the second in the iterative method

We observed that the iterative method still could not track larger motions. But, if we downscaled the images, larger displacements would become easier to track. On the other hand, smaller motions would become more difficult to track as we lose details in the images. To address this problem, we can represent images in multi-scale, and compute flow vectors from coarse to fine scale, and thus conceptualize the Pyramidal Lucas Kanade Optical Flow. The figure below depicts the downsampling of the first frame of the video, giving us the image pyramid representation.

Figure 9: Depiction of various scales of the first frame

For the pyramidal Lucas Kanade we first build pyramid representations, and each level is represented by the downsampling scale. While detection of the position in the next frame, it is first done in the l^{th} level and is then used to initialize the guess in the l-1^{th}, and finally when we reach the highest resolution (the actual resolution) we have an effect from every scale. Thus taking into consideration the small and the large displacements. The optical flow vector obtained as a result of this algorithm is shown below.

Figure 10: Optical flow vectors from the first frame to the second in the pyramidal method

3.3 The Hungarian Algorithm

The Hungarian algorithm, also known as Kuhn-Munkres algorithm, can associate an obstacle from one frame to another, based on a score. We have many scores we can think of including IOU (Intersection Over Union) (meaning that if the bounding box is overlapping the previous one, it's probably the same), shape score (if the shape or size did not vary too much during two consecutive frames; the score increases), convolution cost (we could run a CNN on the bounding box and compare this result with the one from a frame ago, and if the convolutional features are the same, then it means the objects looks the same; or else if there is a partial occlusion, the convolutional features will stay partly the same and association will remain).

3.3.1 Kalman Filter

Kalman Filters are very popular for tracking obstacles and predicting current and future positions. It is used in all sort of robots, drones, self-flying planes, self-driving cars, multi-sensor fusion, etc. A Kalman Filter is used on every bounding box, so it comes after a box has been matched. These bounding boxes were obtained by using the popular algorithm YOLO (You Only Look Once). When the association is made, predict and update functions are called. These functions implement the math of Kalman Filters composed of formulas for determining state mean and covariance. There are two steps for a Kalman Filter to work : prediction and update. Prediction will predict future positions, update will correct them and enhance the way we predict by changing uncertainty. With time, a Kalman Filter gets better and better to converge. For example at time t=0, we have a measurement of 3 bounding boxes. The Hungarian Algorithm defines them at three new detections. We therefore only have 3 detections in our system. For each box, we initialize Kalman Matrices with coordinates of the bounding boxes. Then, at time t=1, we have three bounding boxes, of the same object. The Hungarian Algorithm matches them with the three former boxes and we can start calling predict and update. We predict the actual bounding boxes at time t from the bounding boxes at time t-1 and then update our prediction with the measurement at time t. Prediction phase is matrix multiplication that will tell us the position of our bounding box at time t based on its position at time t-1. For this we define a state transition matrix and a process covariance matrix. Then the update phase is a correction phase.

4 Results

We obtained the best result from the Kalman filtering method which employed the bounding boxes from the YOLO algorithm. We purposefully do not report the mean IoU (Intersection over Union) over all the frames because we feel it is a highly misleading metric for this task. As an example in the Kalman Filtering method in most of the frames the overlap was good but in a few frames where the bounding box was near the boundary the predicted bounding box got stuck in that position for some frames thereby significantly decreasing the mean IoU metric over all the frames, despite performing really well in all the other frames. The methods in this report have been presented

in the increasing order of performance observed by us.

Figure 11: Final frame in tracking

5 Conclusion

In this project we explored various approaches and added our own twist to them to achieve better results. Research has been actively going on in this field owing to its immense applications. Here we only track one person (i.e. one object). New approaches are coming up which can even track multiple objects. Multi-object tracking is becoming increasingly useful to automate processes as simple as taking attendance in a class. Deep Learning approaches are slowly taking over the conventional methods, but only with their help!