Report on Assignment 3

Rahul Bangar, Gururaj Mulay, Tariqul Sifat

March 22, 2017

Abstract

The purpose of this assignment is create a system to detect, and track moving objects from a still camera. To this end, we used MOG for detecting the foreground objects, along with MOSSE to track these across the entire video.

1 Introduction

Our system has two main components: detection of moving objects in a scene, and tracking these objects across the image sequence.

One of the ways in which foreground can be detected is by employing background subtraction. The idea is that we have a reference background model, which we can subtract from any given frame to get the pixels that are different enough (specified by thresholding bounds) from the background, and hence, most probably part of the foreground. In practice, it is not enough to choose any single frame as a reference background model, since there can be illumination changes along with jitter that cause false positives when detecting the foreground.

There are a number of approaches of varying complexity as to how to model the reference background model. On one hand, we have the Mixture of Gaussians (MOG) [4] model, which employs, as the name implies, a mixture of 3-5 Gaussians to model each background pixel. On the other hand, we have VIBE [5], a statistical model of background that does not impose any prior distribution whatsoever on the background pixels, and works in a stochastic manner.

The second component of our system is tracking the objects across the image sequence. Again, a number of alternatives exist: correlating the object template against the entire frame, applying Kalman filtering to model object motion across frames to narrow our correlation window, and MOSSE[2]. MOSSE is an attractive alternative, since it is fast, accurate, and more importantly, doesn't require training beforehand. For our system, we chose MOSSE for these reasons.

2 Description

A detailed description of the system follows:

2.1 Components

The system is comprised of the following components:

2.1.1 MOG

We use the MOG[4] implementation found in the OpenCV[1] library. Our parameters for the MOG are,

$$\texttt{history} = 500, \texttt{Number of Gaussian Distributions} = 5, \texttt{Background Ratio} = 0.5, \texttt{Noise Sigma} = 1$$

2.1.2 MOSSE

We use the implementation of MOSSE[2] found in the OpenCV[1] repository as sample code. The code uses the same techniques described in the MOSSE[2] paper.

2.2 Methodology

Our combination of the two above mentioned component can be used to track moving objects in real time. The methodology of the system is as follows:

2.2.1 Filtering

The binary image that we get from the MOG is passed through an opening filter (5,5) to remove noise (insignificant foreground pixels). The output is then passed through an erosion filter (7,1) that is useful for removing horizontal shadows. We use a configurable parameter to define the number of iterations we run the erosion filter to remove shadows. Then the output of applying erosion filter is passed through a Gaussian mask of size (99,99). Then we threshold the blurred image to get a binary image that represents our post-processed foreground frame. Combining these filter operations, we obtain a foreground mask that should be unperturbed by the random noise in the scene.

2.2.2 Finding windows

It can be seen that not all foreground regions are significant enough to be tracked. So, we process it to find the contours, which are thresholded by a lower bound of contour area to find significant foreground regions. This allows us to modify our approach for different scenes, while avoiding noise at the same time.

2.2.3 Tracking

The challenge of combining the two components (Foreground detection and Tracking) is to figure out when to introduce a new tracker using the windows that we find from the foreground detection model. We get a list of windows from our foreground detection model. For every window, we have to decide if this window should introduce a new tracking window, if it should be ignored because it's already being tracked by MOSSE or if this window is already (mostly) covered by one or more MOSSE trackers. Here is the pseudo code of our algorithm for introducing MOSSE tracker in the video.

```
for each frame:
    Update all existing MOSSE trackers;
    Get a list of foreground windows from MOG;
    for each foreground window:
        if the window is already mostly covered by the union of
                existing MOSSE trackers:
            ignore it, because the foreground window is probably
            an existing tracker or a combination of them
        else if the window is overlapping with another tracker:
            if the window size is significantly different from
            the tracker size:
                update(reinitialize) the tracker with
                the size of the foreground window
            else:
                ignore the foreground window
        else:
            Add the foreground window as a new MOSSE tracker
```

3 Evaluation

There are a wide variety of evaluation methods available for object detection and tracking systems, however, we primarily focused on the standard metric – 'frame–based evaluation metric' described

		Ground Truth		
		Positive	Negative	Total
SUT Output	Positive	$TP = 110$	$FP = 0$	$TP + FP = 110$
	Negative	$FN = 46$	$TN = 63$	$FN + TN = 109$
	Total	$TP + FN = 156$	$FP + TN = 63$	

Table 1: Confusion Matrix Video example1.mov with overlapThreshold=0.25

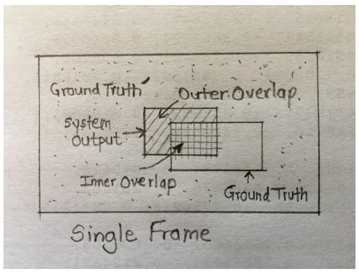


Figure 1: Overlap-based method graphic

in [6] and [7] and one another method which calculates the area of overlap between the systems actual output windows and the ground truth windows on per frame basis. We evaluated the example1 video under both the methods. The **Ground Truth** was generated using the VATIC tool for labeling the dataset videos provided by UCI [8]. The csv file for ground truth of video example1.mov is submitted along the code as `Ground_truth_OBJ_ID.csv` wherein we have ground truth labels (i.e., windows) for every object stored in individual csv files. Here `OBJ_ID` refers to the individual objects in the video. Now we describe both the methods in details.

3.1 Method 1: Frame-based Metric

For MOG and MOSSE method on example1.mov, we compared every video frame in the Ground Truth (GT) with the corresponding frame from the output of our System Under Test (SUT) in terms of the bounding box (window). There are two chief way to compare two frames [6]:

a) Objects are matched from two frames based on the overlap area of two object windows reported at each frame. This is the method that we used for our testing. The overlap is simply the intersection of two rectangular frames (SUT and GT) with respect to the original area of the GT label window. The overlap threshold selected for a successful match is kept as **0.25** for example1.mov video to consider as a successful detection.

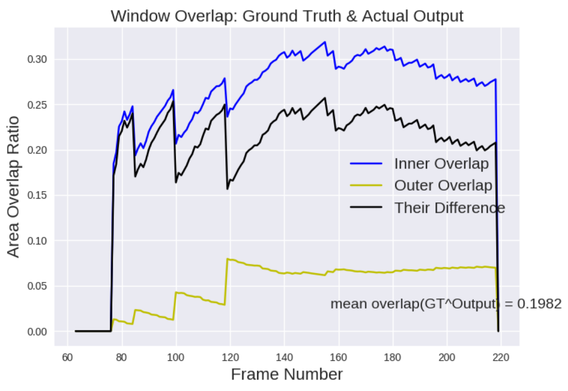
b) The second method is to match the objects based on the distance between the centroids of an object's bounding boxes from GT and SUT. Alternatively, we check if at least one of the centroids lies within the bounding box of the other [7] for it to be considered as successful detection.

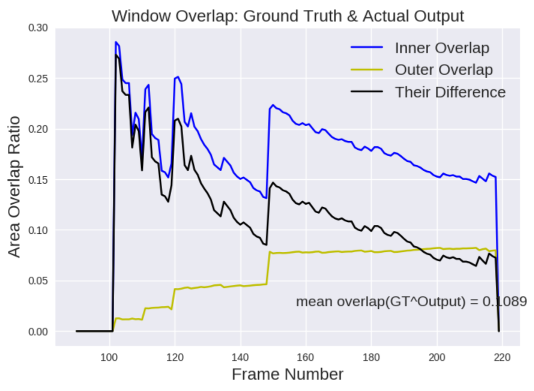
Evaluation: We use the standard terms used in [7] that define TP, TN, FP, FN. The definition of success per frame mentions frame is considered as successful in detection if at least one bounding box from GT overlaps with (with a certain threshold with the SUT's bounding box. Using these definitions, we have the confusion matrix as in table 1. Using the standard definitions we have **Detection Rate=0.7051, FAR=0, FPR=0, FNR=0.2948, Precision=1.0, Recall=0.7051.**

3.2 Method 2: window overlap-based metric

Here, for each frame we calculate individually for every object, the overlap of GT and SUT output. **Inner Overlap Ratio** (IOR) is (GT ∩ SUT)/GT and the **Outer Overlap Ratio** (OOR) is (GT' ∩ SUT-(GT ∩ SUT)))/GT' as shown in figure 1. So the difference between IOR and OOR gives a metric to calculate how much does the SUT overlaps GT. OOR penalizes if the SUT window lies outside GT. Graphs in table 2, depict the IOR in blue, OOR in red, and difference in black. For example1.mov, the mean IOR-OOR is 0.2 and 0.1 for truck0 and truck1.

Table 2: Window Overlap (GT Intersection SUT): for all frames: truck0, truck1





4 Conclusion

In this assignment, we tried MOG in combination with MOSSE along with a morphological filtering approach to detect and tracking moving objects in a video. From the gathered results, it is easy to see that motion tracking is not a trivial problem, especially in the absence of object classification. Using morphological operations to filter the foreground is one of the approaches, but we believe that it could be supplanted by something better, especially if we augment our system with object classification. As an example, shadows pose a major problem in detection, especially in the particular video that we tested. It would be useful to try shadow removal during detection before we track the objects, which should increase the detection rate.

References

[1] Itseez, Open Source Computer Vision Library, 2015, https://github.com/itseez/opencv

[2] David S., Bolme J., Ross Beveridge, Bruce A. Draper, Yui Man Lui, Visual Object Tracking using Adaptive Correlation Filters, Computer Science Department, Colorado State University, Fort Collins, CO 80521, USA

[3] Open Source Computer Vision Library Samples (Python), Steven Puttemans, 2016, https://github.com/opencv/opencv/blob/master/samples/python/mosse.py

[4] Stauffer C., Grimson W., Adaptive background mixture models for real-time tracking, The Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139

[5] O. Barnich and M. Van Droogenbroeck. ViBe: A universal background subtraction algorithm for video sequences. In IEEE Transactions on Image Processing, 20(6):1709-1724, June 2011.

[6] A. Godil, et al., Performance Metrics for Evaluating Object and Human Detection and Tracking Systems, NISTIR 7972, July 2014

[7] F.Bashir, Performance Evaluation of Object Detection & Tracking Systems, CVPR, June '06

[8] http://web.mit.edu/vondrick/vatic/ + https://dbolkensteyn.github.io/vatic.js/