COL780 : Computer Vision
Background Subtraction

Suyash Agrawal
2015CS10262

September 18, 2017

1 Introduction

In this assignment we had to implement Background Subtraction using mixture of Gaussian model. This is the base model proposed by Strauffer and Grimson in their paper.
Background subtraction using gaussian mixture model is done by fitting k-gaussians on each pixel, which model the pixel value distribution. These gaussians are fitted using E.M. algorithm and their online approximation is used for performance.

2 Implementation

First attempt to implement background subtraction was done in python using loops. The algorithm implemented was taken from [?] and the results were found to be poor. The main drawback was that the speed was very slow and it took approx. 30 seconds to process just one 600x700 frame.
Further attempts were made to convert this program to use only matrix operations and no loops as matrix operations are highly optimized in numpy library. The result was that the speed was now good but the code became quite complex. This resulted in hard debugging sessions and less flexibility in experimentation.

Finally, the code was written in C++, so that loops could be used without performance degradation and this resulted in easy experimentation and debugging sessions. I tested the algorithm for various values of hyper-parameters, whose details are given in the next section.

3 Observations and Results

The testing of the program was done for various test videos like: remote controlled car video, people walking video, webcam video, simple animated video. The results were also compared to the native implementation in OpenCV library.
The program basically had these hyperparameters:

- K - The number of gaussians to model each pixel.
- α - The learning rate of the algorithm
- T - The term deciding the number of gaussians representing background
- std_dev- The initial standard deviation of each gaussian.
- mean - The initial mean of each gaussian

The observations after experimentation were:

- The initial std_dev should be kept high. Small values of std_dev resulted in false classification of background pixels as foreground pixels.
- The initial mean did not affect the performance much. In my implementation I set it to 0 initially. This might be because mean can be very easily modified by algorithm to any value and thus having little impact on performance.
- T is a very important factor. Setting its value to low (e.g. 0.1) resulted in almost all of the picture to be classified as foregroung, while setting to high values(e.g. 0.9) resulted in very little foreground points. The value that I got good results with were around 06-0.8 .
- α (Learning Rate) is also important factor as it determined how much time steps the algorithm took to classify a pixel as background. Large value led to very sparse and disconnected foreground (as the pixels were quickly merged to background), while low values resulted in unnatural trails and disturbance in the output.
- K (number of gaussians) were a key factor for the performance of the algorithm. Low values of K (like 1-2) resulted in very poor performance as there were not enough gaussians to model complex environment like people walking in street, while setting large values (like 6-7) also result in degraded performance. The optimal values were around 3 and I got the best result by setting the parameter 3.

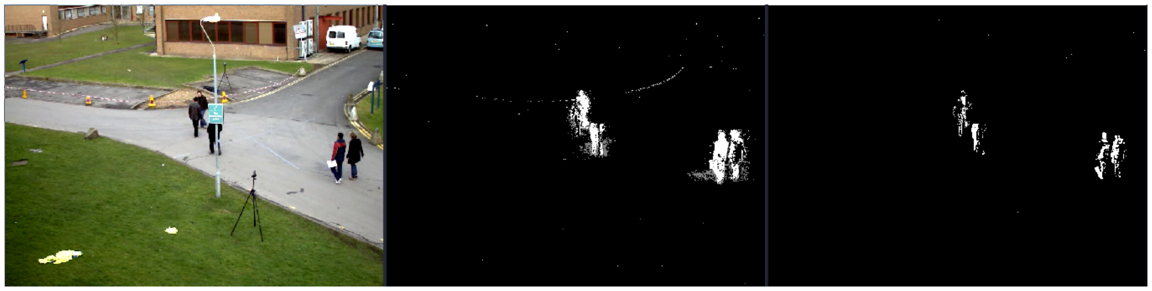

Figure 1: Result with 3 gaussians. (middle is opencv result)

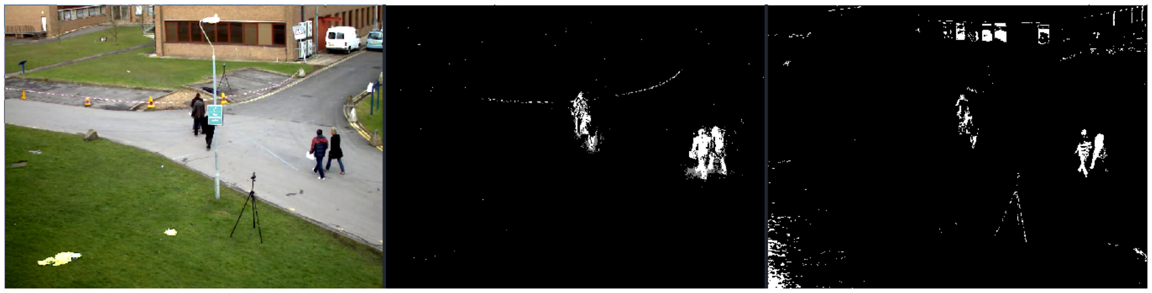

Figure 2: Result with 4 gaussians. (middle is opencv result)

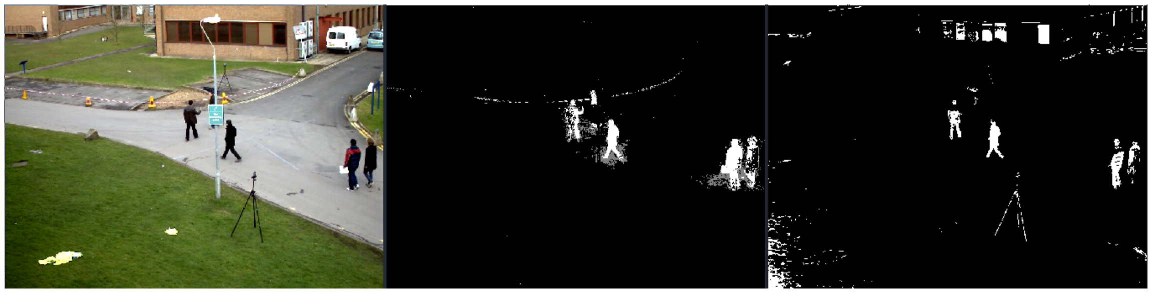

Figure 3: Result with 5 gaussians. (middle is opencv result)

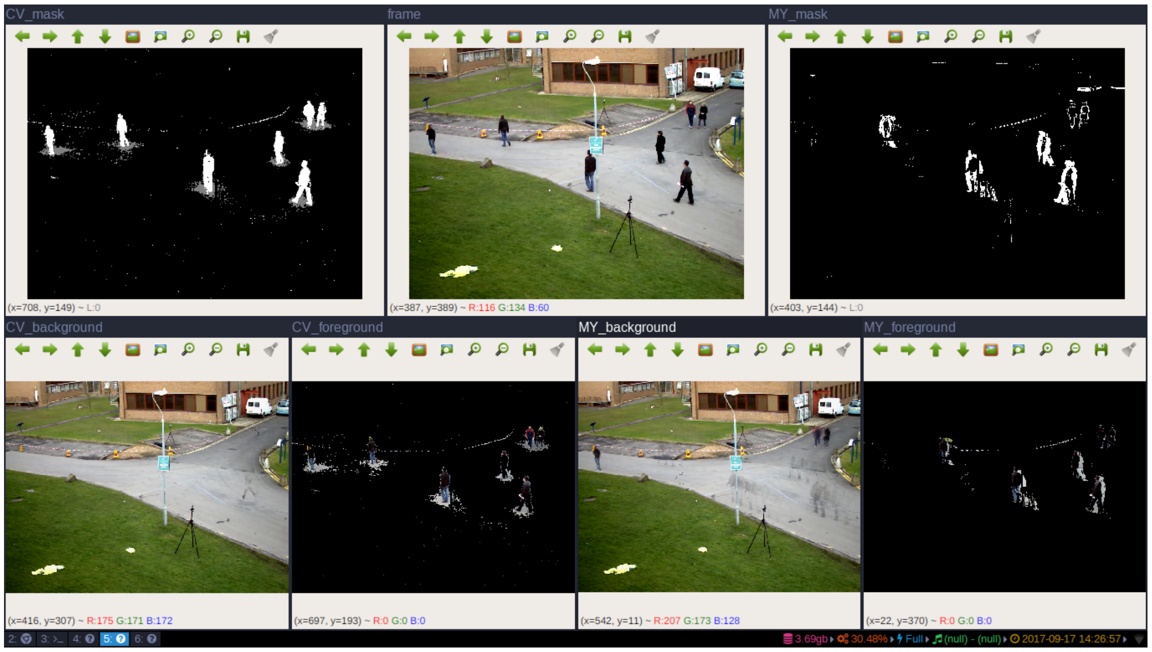
CV_mask
frame
MY_mask
(x=708, y=149) ~ L:0
(x=387, y=389) ~ R:116 G:134 B:60
(x=403, y=144) ~ L:0
CV_background
CV_foreground
MY_background
MY_foreground
(x=416, y=307) ~ R:175 G:171 B:172
(x=697, y=193) ~ R:0 G:0 B:0
(x=542, y=11) ~ R:207 G:173 B:128
(x=22, y=370) ~ R:0 G:0 B:0
3.69gb 30.48% Full (null) - (null) 2017-09-17 14:26:57

Figure 4: Result with pedestrian video in python implementation

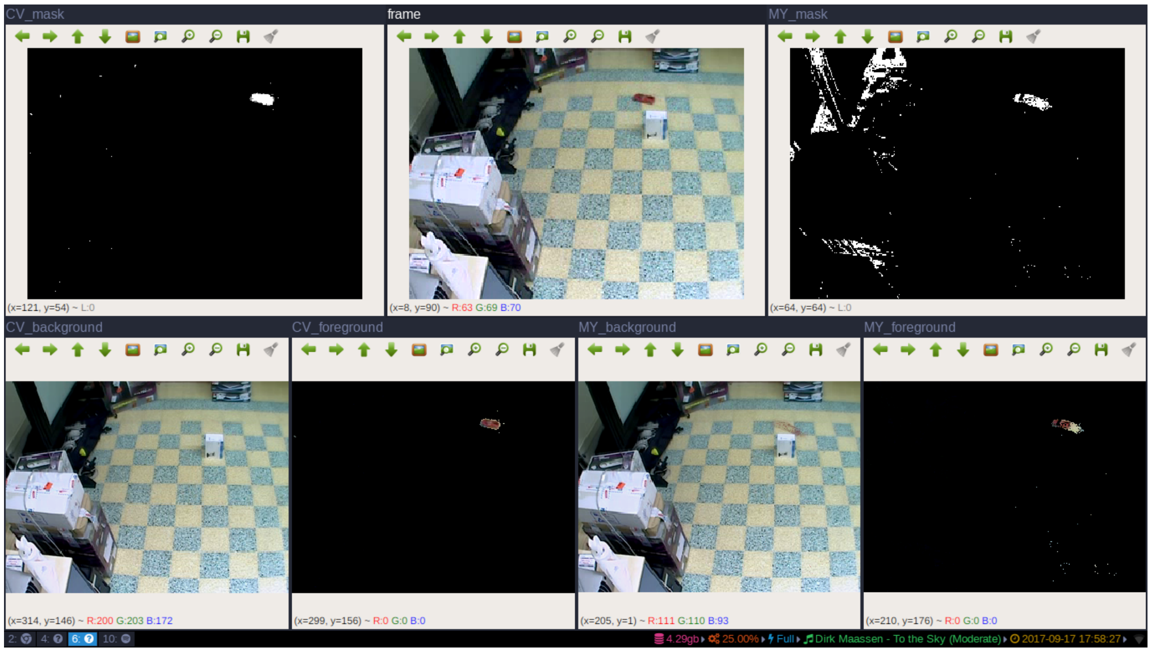
CV_mask
frame
MY_mask
(x=121, y=54) ~ L:0
(x=8, y=90) ~ R:63 G:69 B:70
(x=64, y=64) ~ L:0
CV_background
CV_foreground
MY_background
MY_foreground
(x=314, y=146) ~ R:200 G:203 B:172
(x=299, y=156) ~ R:0 G:0 B:0
(x=205, y=1) ~ R:111 G:110 B:93
(x=210, y=176) ~ R:0 G:0 B:0
4.29gb 25.00% Full Dirk Maassen - To the Sky (Moderate) 2017-09-17 17:58:27

Figure 5: Result with car video in python implementation

4 Conclusion

I also found that OpenCV implementation was highly optimized, both in terms of speed and quality, as compared to this algorithm. After a bit of research, I found that OpenCV's algorithm did not use a fixed number of gaussian for each pixel and decided it dynamically along with various other optimizations. Also, it did all the operations in matrix form which gives a huge speedup (as can be seen by me during python implementation).

In my experiments I found the results of OpenCV to be superior to mine. Though there were cases when OpenCV's output was fuzzier and more delocalized than mine, but in general OpenCV results were very dense while my results, though very localized, were a bit sparse and there were sometimes holes inside the objects.