COP290 Assignment 1 Report

Mihir Meena (2019CS10370)
Nishant Kumar (2019CS50586)

March 2021

1 Introduction

This is the project report for assignment 1 on Traffic Density estimation using OpenCV functions done under the course COP290. Three sections, namely, **Metrics, Methods** and **Trade-Off analysis** are discussed ahead.

2 Metrics

Utility is taken to be the mean error generated when using different methods. Error is calculated by taking difference of queue density value for a particular frame with queue density value for that same frame in the baseline method. Taking mean of those errors across all the frames is taken as utility.
Runtime is the time taken (in seconds) by a particular method to process the complete video.

3 Methods

3.1 Method 1

Method 1 involves sub-sampling of frames i.e skipping some, say x frames while processing the video and the queue density values for the intermediate frame is chosen accordingly. For example, if N frames are already processed then next x frames will be skipped and will not be processed and all the intermediate frames will get the queue density value of the (N+x)th frame. Parameter clearly then is x for this method. Runtime and utility values were generated for x=1 to x=10 giving 10 points to plot the graph.

3.2 Method 2

Method 2 involves decreasing the resolution of the frame on which background subtraction is to be applied. Homography is first applied on the frame extracted from the video to obtain the cropped frame with corrected angle. Resolution of this frame is then decreased and subsequently BGSub is applied. Parameter in this case is X and Y, the dimensions of the frame. To obtain runtime and utility values both X and Y were divided by 2 in each iteration and total 7 iteration were executed giving 7 points to plot the graph.

3.3 Method 3

Method 3 involves multithreading with the help of pthreads. Each thread is given a part of the frame which is obtained after splitting the frame appropriately. Parameter in this case is the no. of splits which is same as no. of threads as each thread gets a single split to process. Runtime and utility values were generated for x=1 to x=10 giving 10 points to plot the graph.

3.4 Method 4

Method 4 also involves multithreading. But in this case consecutive frames are given to different threads to process. Parameter in this case is no. of threads. Runtime and utility values were generated for x=1 to x=10 giving 10 points to plot the graph.

4 Trade-off analysis

In this section all the methods mentioned above are analysed for any relation between **Runtime**, **Utility** and **Parameters**. In some cases the results are as expected whereas in others the result are not that obvious. Possible explanations for such cases are provided.

4.1 Method 1

In this method x frames are skipped i.e are not processed, x being the parameter. Clearly the runtime should decrease as we keep skipping more no. of frames in each iteration. So the graph of Runtime v/s Parameter should be of decreasing nature. Also as we increase the no. of frames skipped the mean error is bound to increase which indicates that the graph of Utility v/s Parameter should be of increasing nature. From the above two observations, the graph of Utility v/s Runtime should be of decreasing nature as less runtime implies greater value of parameter which implies greater value of error. Table 1 and Figure 1 and 2 are relevant to method 1 which are in agreement to the above conclusions.

x	Runtime	Utility
1	47.333	0.000000
2	34.803	0.001541
3	29.889	0.002647
4	28.169	0.004346
5	26.122	0.005568
6	25.097	0.006726
7	24.742	0.008215
8	24.178	0.009428
9	24.026	0.010711
10	23.799	0.012145

Table 1: Runtime and Utility values for method 1

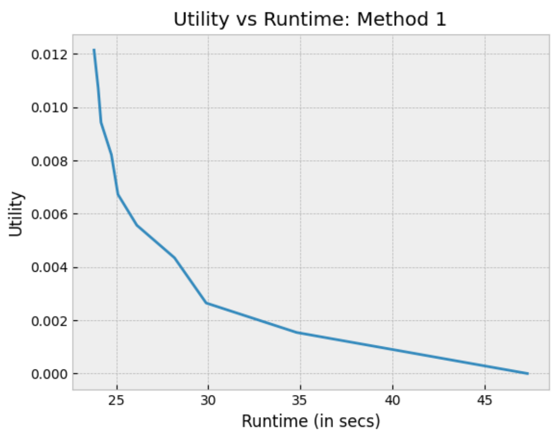

Figure 1: Utility v/s Runtime for method 1

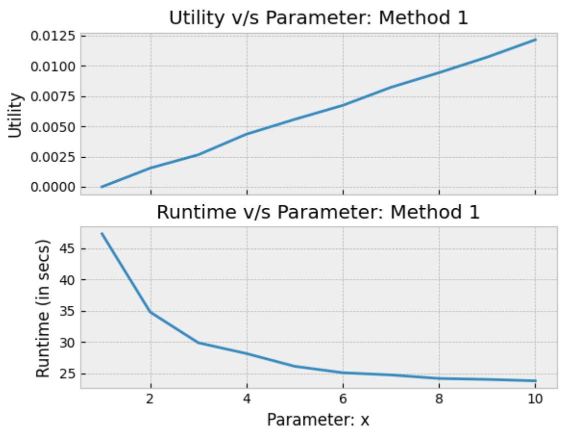

Figure 2: Utility and Runtime v/s Parameter for method 1

4.2 Method 2

This method reduces the resolution of the frame on which background subtraction is to be applied thus reducing the no. of pixels to be processed. In each iteration both the dimensions X and Y are halved thus reducing the total count of pixels by a factor of 4. As the total no. of pixels are reducing in each iteration the runtime should also decrease. So the graph of Runtime v/s Parameter should be of decreasing nature. The mean error will increase as the resolution is decreased since the queue density computed will not be accurate due to loss of information while reducing the resolution of the frame. So the graph of Utility v/s Parameter should be of increasing nature. From the above two observations, the graph of Utility v/s Runtime should be of decreasing nature as less runtime implies greater value of parameter which implies greater value of error. Table 2 and Figure 3 and 4 are relevant to method 2 which are in agreement to the above conclusions.

Resolution(X*Y)	Runtime	Utility
200*500	48.525	0.000000
100*250	45.484	0.003874
50*125	42.731	0.004395
25*62	41.016	0.004113
12*31	40.16	0.007128
6*15	39.929	0.021899
3*7	39.881	0.034690

Table 2: Runtime and Utility values for method 2

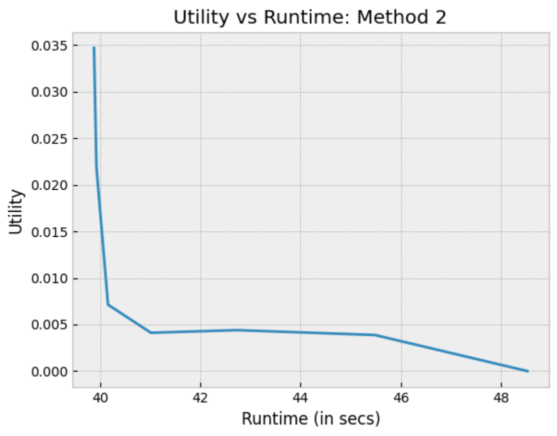

Figure 3: Utility v/s Runtime for method 2

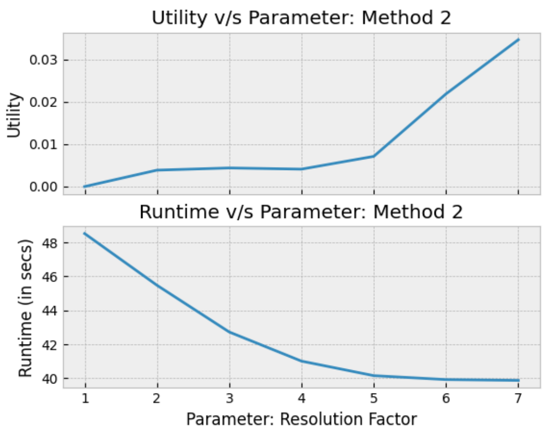

Figure 4: Utility and Runtime v/s Parameter for method 2

4.3 Method 3

This method involves use of multithreading (using pthreads). The frame which is to be processed is first spilt and then different threads carry out background subtraction on those part in parallel. The queue density calculated in this way for each frame will be equal to the queue density value in baseline for that same frame. So error in this method is zero independent of runtime or parameter. Clearly then the graph of Utility v/s Runtime and Utility v/s Parameter should be a line parallel to x-axis with utility value being zero which can be seen in Fig. 5 and 6. Now as the number of threads are increasing the work should be divided among them and the overall runtime should decrease. But as can be seen clearly from the graph of Runtime v/s Parameter in Fig. 6 this is not the case rather the runtime increases for greater values of parameter. A possible explanation for this observation could be that the time required for splitting a frame is significant and is compensating for the decrease in runtime due to increase in count of threads and even more than compensating for greater values of threads which require more splits.

No. of threads	Runtime	Utility
1	47.972	0.000000
2	47.752	0.000000
3	47.575	0.000000
4	47.966	0.000000
5	48.523	0.000000
6	48.64	0.000000
7	50.122	0.000000
8	49.192	0.000000
9	50.214	0.000000
10	49.559	0.000000

Table 3: Runtime and Utility values for method 3

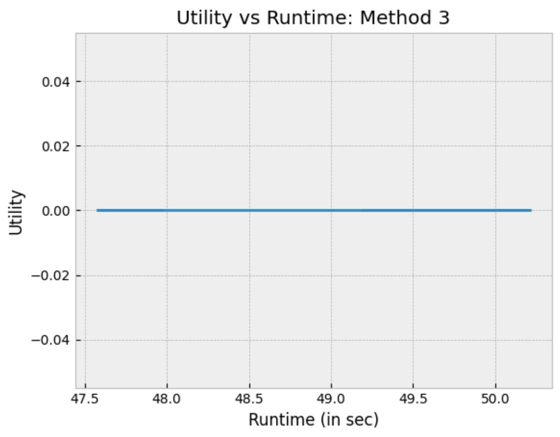

Figure 5: Utility v/s Runtime for method 3

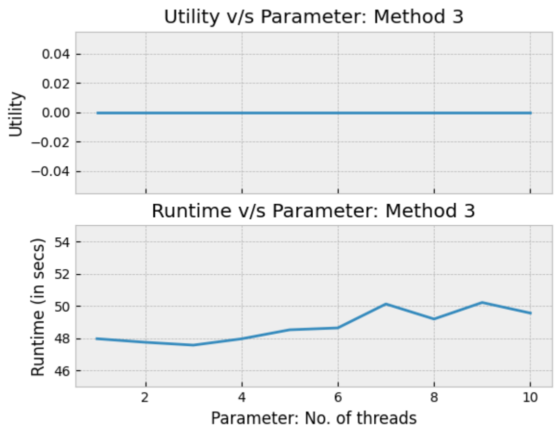

Figure 6: Utility and Runtime v/s Parameter for method 3

4.4 Method 4

This method involves multithreading as well. But the difference from method 3 is of what is passed to a single thread. In this method, consecutive frames are passed to different threads. Therefore, each thread calculates queue density for a single frame. In contrast to method 3, there is no splitting required in this method. Also processing a single frame takes a significant amount of time, so running that process in parallel reduces the runtime to quite a good extent which can be seen in Runtime v/s Parameter graph in fig. 8. But the error in this method, though small is non-zero value. This is due to fact that the function that different threads are using is having different instances are running in parallel, so, some particular no. of frames equal to the parameter (no. of threads) are being processed simultaneously. So due to non-synchronization of pthreads sometimes the queue density value is overwritten due to some other instance of the function running in parallel. This induces a slight deviation in the queue density values recorded which is not that significant as nearby frames have almost same queue density values. Also this is error is not increasing with a fast rate unlike method 1 and 2 where for large value of parameter, error v/s parameter graph increases very fast (fig. 2 and 4) rather it saturates for larger values of parameter as can be seen in the Utility v/s Parameter graph (fig. 8).

No. of threads	Runtime	Utility
1	60.132	0.000000
2	55.912	0.001187
3	49.27	0.002472
4	45.864	0.002966
5	42.561	0.003220
6	41.167	0.004162
7	40.047	0.003900
8	38.89	0.004093
9	38.226	0.004243
10	34.188	0.004150

Table 4: Runtime and Utility values for method 4

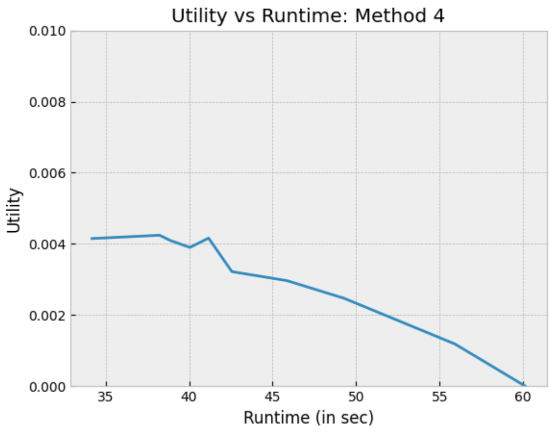

Figure 7: Utility v/s Runtime for method 4

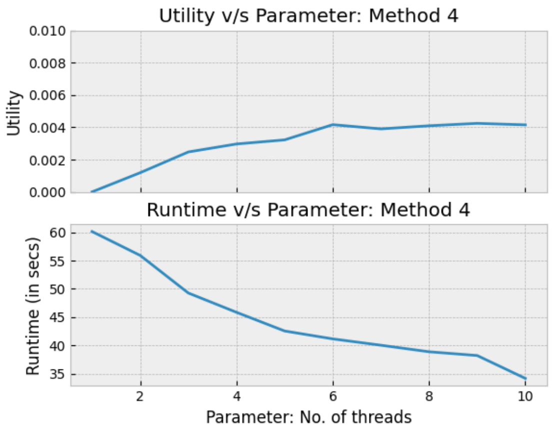

Figure 8: Utility and Runtime v/s Parameter for method 4

5 Conclusion

After analysing all the methods mentioned above i.e method 1 through 4, some conclusions can be drawn from the graphs obtained. In method 1 runtime decreases by increasing the parameter but error increases as well and for higher values of parameter error grows very fast. Same conclusion can be drawn for method 2. So optimum value in this method could be the ones in which error is tolerable and runtime is optimized as well. In method 1 this value could be corresponding to x=4 where runtime is 28.169 and utility is 0.004346. Similarly for method 2 resolution 25*62 could be considered optimum with runtime 41.016 and utility 0.004113. The higher runtime value for method 2 in comparison to method 1 could be down to fact that resizing of a frame takes up a significant amount of time in comparison to skipping frames in method 1.
In method 3 runtime remains almost the same (possible reasons disscused in previous section) and error is zero throughout. So lowest runtime should be the optimum value which is for 3 threads having runtime 47.575. Clearly not better than method 1 and 2.
Method 4 is having the highest value of error for no. of threads=10 which is still approximately same as that of optimum points for method 1 and 2. But runtime value (34.188) is significantly better than optimum point's runtime value for method 2 (41.016) and comparable to method 1 (28.169).
From above discussion it can concluded that in this case Method 1 and 4 are clearly the ones to chose from. Method 1 could be preferred due to its low runtime value but in some other case Method 4 might be better due to its low value of error even for high values of parameter unlike method 1 which has a very fast rate of increase in error at the higher end of parameter values.