TRAFFIC DENSITY ESTIMATION USING OPENCV

R HARI SHANKAR
2019CS10386

DISHANT DHIMAN
2019CS10347

FINAL ANALYSIS REPORT
COURSE : COP290

2020/2021 Semester 2

Acknowledgement

We would like to express our special thanks and gratitude to our professor Dr.Rijurekha Sen , who gave us this wonderful project on ”Traffic Density Estimation”, and also guided us to complete the project .Secondly we would also like to thank all the TA’s in the COP290 course for clearing doubts in time .

R Hari Shankar
Dishant Dhiman

Table of Contents

Introduction

Camera angle correction and frame cropping

We have used the concept of homography using OpenCV to create a C++ program which takes the four points of the image (road) to change perspective such that we get the top view of the road by warping the image with 4 points of rectangle which corrects the angle of camera as if viewing from the top .The cropping of the image is done by using the inbuilt function of OpenCV which allows us to crop the image by providing the starting point and the length and breadth of the needed region. We have implemented abstraction as much as possible so that these functions can be used for other programs as well.

Queue and Dynamic Density estimation

We have used the concept of background subtraction and optical flow across different frames to compute the queue and dynamic densities. For this we have taken the help of angle correction and cropping functions to get a top view of the road from the video file and computed the densities according to the pixel changes in the images. The densities calculated are saved in the file ”out.txt”. A make file is provided to make the job of running the program easier for the user . We have tried to make the code as understandable as possible by adding the comments wherever required.

Utility-Runtime trade-off Analysis

Taking the outputs given in the part(b) of assignment as baseline we will conduct utility vs runtime analysis . Baseline is the method against which we compare other methods/parameters, and see whether we are getting a better or worse trade-off. The utility function is calculated by noting difference in queue per unit time compared to baseline method and then taking the squared or absolute value of the error and take the mean of the value over the whole video to get the utility of the method over the video. Finally we plotted the utility against runtime to analyse which trade-off improves or degrades.

The methods/parameters for conducting trade-off analysis are :

1. Sub-sampling frames – processing every x frame i.e. process frame N and then frame N+x, and for all intermediate frames just use the value obtained for N - total processing time will reduce, but utility might decrease as intermediate frames values might differ from baseline. Parameter for this method is x, how many frames you drop.
2. Reduce resolution for each frame. Lower resolution frames might be processed faster, but having higher errors. Parameter can be resolution XxY.
3. Split work spatially across threads (application level pthreads) by giving each thread part of a frame to process. Parameter can be number of splits i.e. number of threads, if each thread gets one split. You might need to take care of boundary pixels that different threads process for utility.
4. Split work temporally across threads (application level pthreads), by giving consecutive frames to different threads for processing. Parameter can be number of threads.

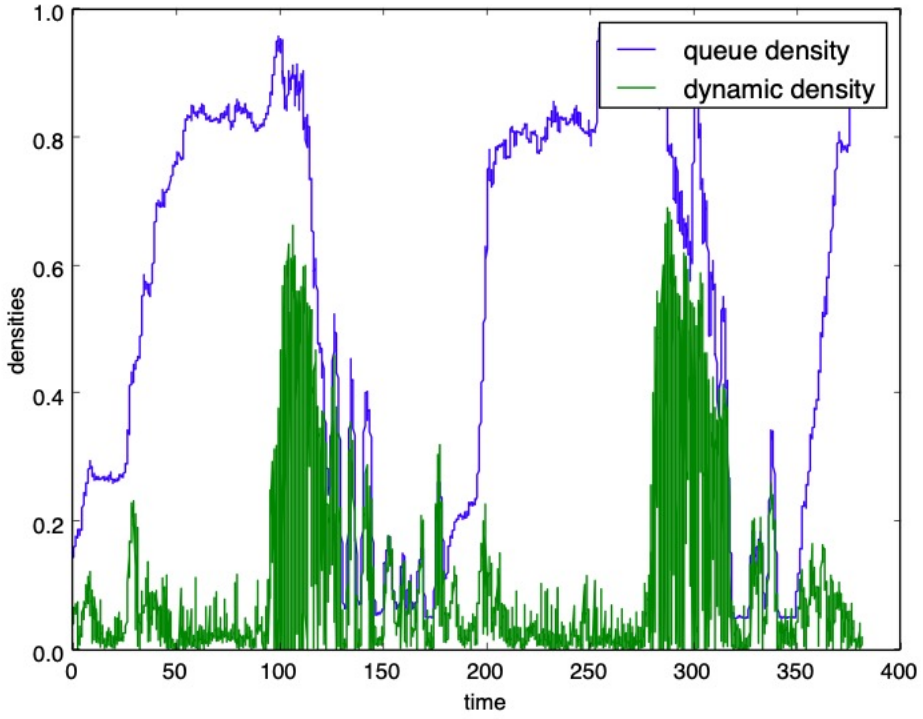

Figure 1.1: Graph of Baseline Method

Understanding and analyzing trade-offs in software design

2.1 Metrics

The metrics used for the trade-off analysis are different for each method . The utility and runtime metrics for the methods are :

1. **Method 1 :** As this method implements sub-sampling of frames we use the *parameter* 'x' which is the number of frames dropped such that we process frame N and then process frame N+x. For *utility* we use the mean of absolute error per frame compared to **baseline (x=0)** . The runtime is computed in seconds using the inbuilt system clock from the chrono library in cpp.

2. **Method 2 :** As this method changes the resolution of each frame in the video we use *parameter* XxY which is the changed resolution of the frame .For *utility* we use the mean of absolute error per frame compared to **baseline (328*778)** .The runtime for different parameter values is computed in seconds using the inbuilt system clock from the chrono library in cpp.

3. **Method 3 :** The metric used for computing the *utility* is absolute difference method compared to **baseline(cpu cores = 1)**.We will use number of threads as *parameter*. The runtime is computed in seconds using chronos library.

4. **Method 4 :** We have to split the work across threads by giving consecutive frames to different threads for processing . We will use the number of threads as *parameter* .We will use **1 thread as baseline** for this method . The runtime, average CPU usage per thread and sum of absolute error per frame compared to baseline will be taken as metrics for the *utility* function .

2.2 Methods

The methods that we could implement along with its distinct parameters are given in this section .

2.2.1 Method-1

In this method we have taken x as the parameter so we vary the value of x from 0 to 30 and get the utility value by getting the mean of absolute difference from the baseline (when n=0) .These values are then used to plot the graphs for trade-off analysis. We also take the runtime values for each parameter for analysis.

2.2.2 Method-2

In this method we have taken the resolution X*Y as parameter so we find the density values and use it to get the utility for distinct parameter values and use it to plot the graphs for trade-off analysis. We also take the runtime values for each parameter for analysis.

2.2.3 Method-3

In this method, a frame extracted at a certain point of time is divided in parts equal to the number of cpu cores engaged. This division is done by copying the frame into smaller matrices. The empty image is cropped as well. Queue density is obtained for all the parts and averaged out.

2.2.4 Method-4

In this method we have passed consecutive frames to different threads and we have taken the number of threads used as parameter while calculating queue density . We use the utility , runtime and CPU measure along with parameter for trade-off analysis.

Trade-off Analysis

3.1 Method-1

From figure 3.1 it is clear that as the parameter x increases (no of frames to be skipped) then the runtime of program decreases , this is due to the fact that more the number of frames we skip less will be the number of computation which implies less runtime .
From the utility-parameter graph we also infer that as the parameter increases the the utility follows a decreasing trend (with some exceptions caused due to significantly large error values). This decreasing trend in utility is because we are using less frames for finding the queue density leading to less average error
The utility vs runtime graph shows how the utility function first increases with increasing runtime, the reason for this can be directly inferred from the above arguments.

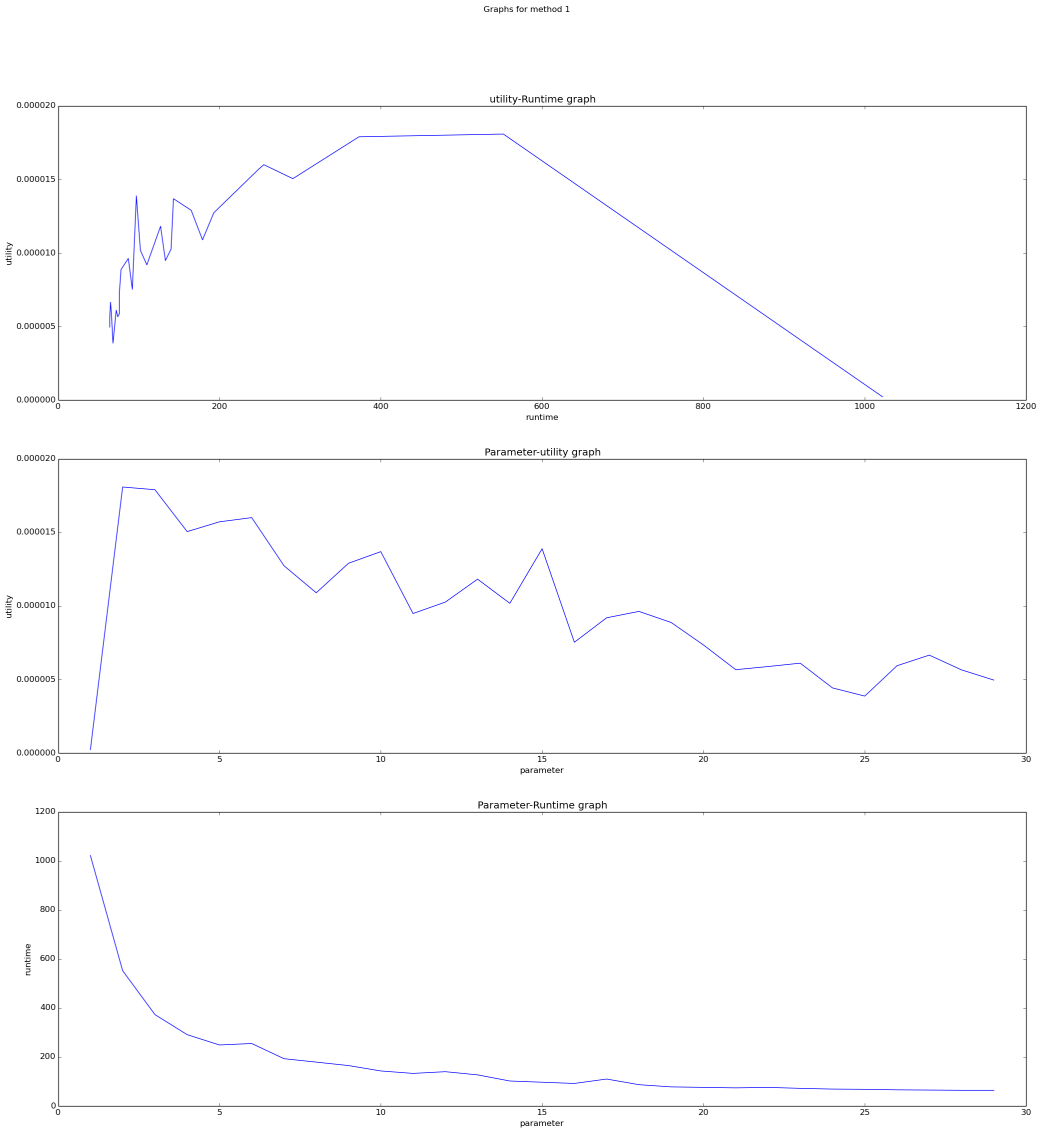

Figure 3.1: Trade-off analysis of method-1

3.2 Method-2

From Figure 3.2-3.3 we find out that the when the the resolution of frame (parameter) is less the queue and dynamic density values increases as compared to baseline .This is due to the fact that when we reduce the resolution ,percentage of pixel changes increases as pixels are binned together while decreasing resolution as a result background subtraction and optical flow values increases resulting in more density for lower resolution.

Figure 3.4 gives the trade-off analysis for queue density and it shows that as parameter values increases the runtime also increases ,this is because when we have small parameter value the no of pixel to be compared are less for background subtraction is less thus taking less time to compute.

The utility-parameter graph shows how the utility(error) decreases with increasing parameter(resolution) this is due to the fact that larger resolution image has crisper details leading to less error compared to baseline.

The utility decreases with increasing runtime which can be inferred directly from the above reasoning.

Figure 3.5 shows the trade-off analysis for dynamic density which has result similar to queue density with slight irregularity for the 164*389 parameter case.

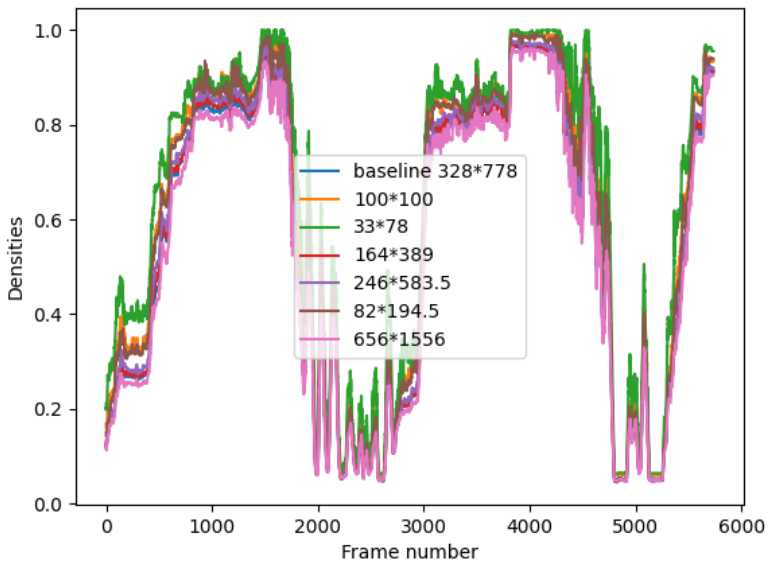

Figure 3.2: Variation of queue density for different parameter values

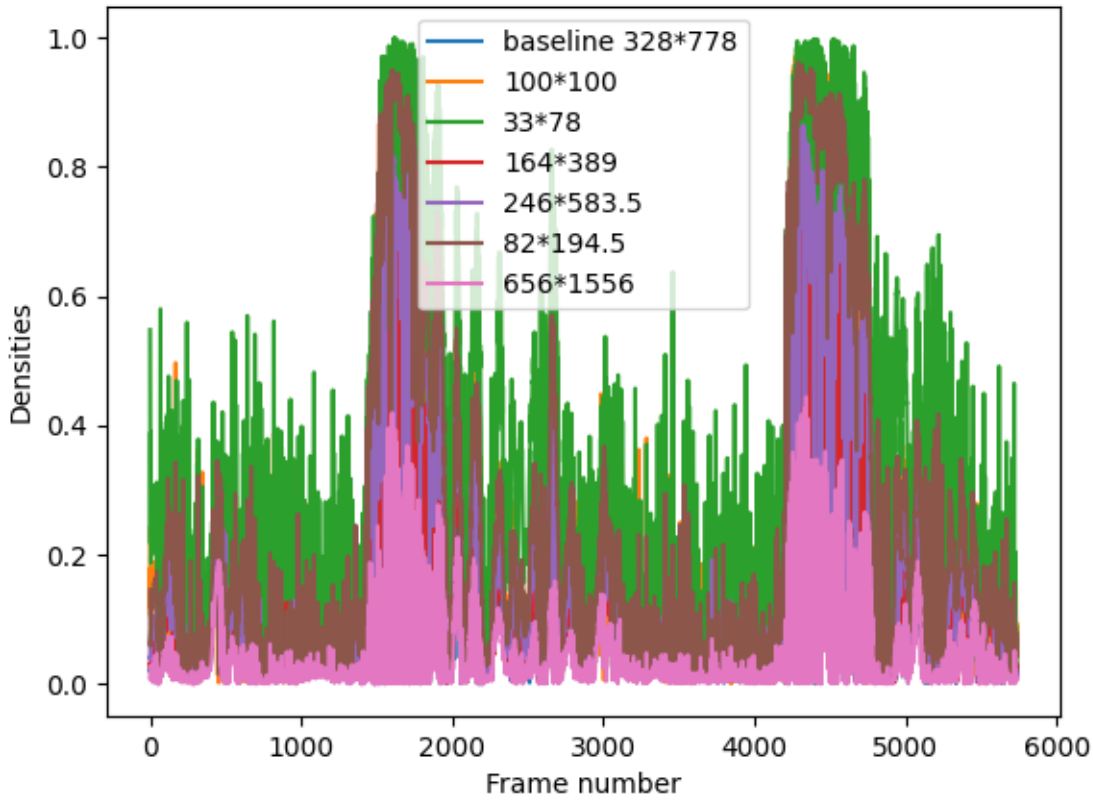

Figure 3.3: Variation of dynamic density for different parameter value

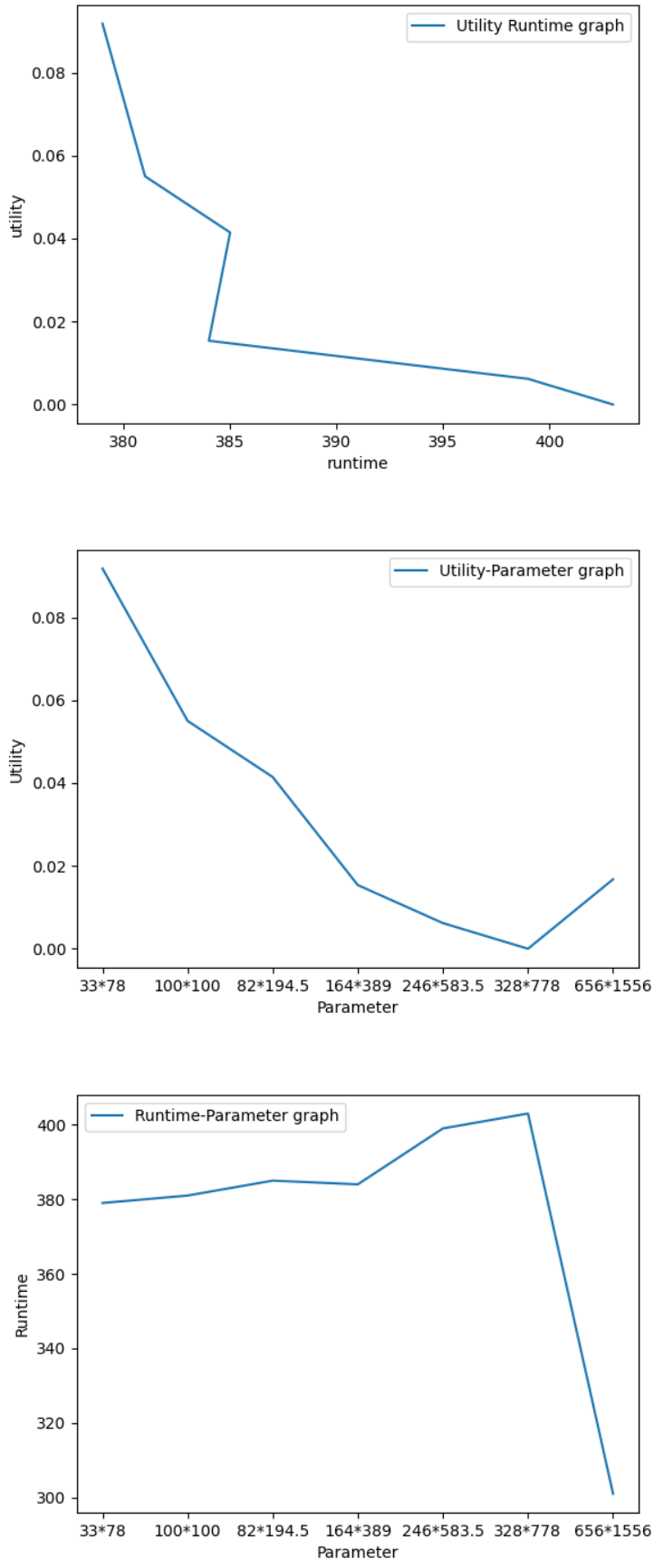

Figure 3.4: Trade-off analysis of queue density method-2

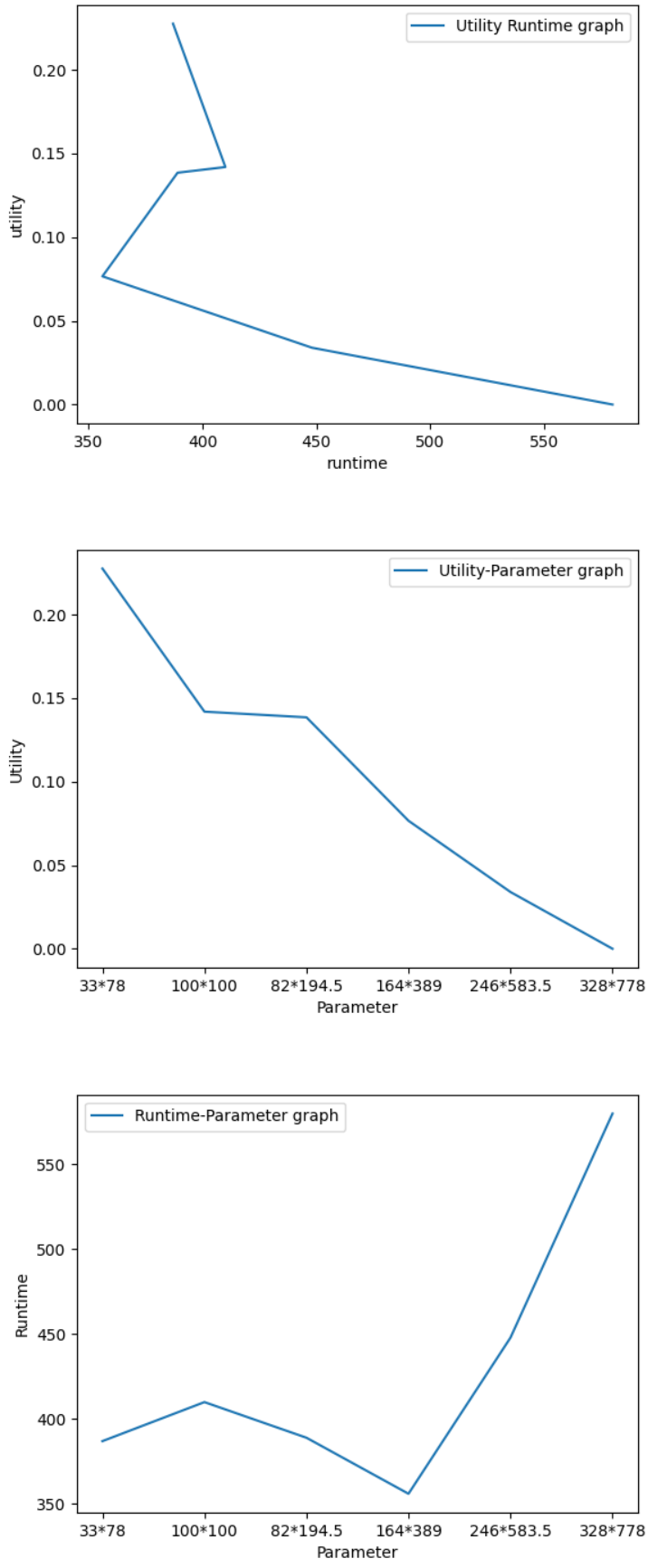

Figure 3.5: Trade-off analysis of dynamic density method-2

3.3 Method-3

From the queue density plots, it is evident that the accuracy to determine density is poor in case of higher number of cores, but that too with acceptable amount of error. The utility/error was constantly increasing with the number of splits as expected because the corner pixels of the split images are in different environment.

Also, the runtime is increasing with the parameter. The reason might be that the overhead expenses to create threads, split two images may be greater than the benefits provided by using multi-threads.

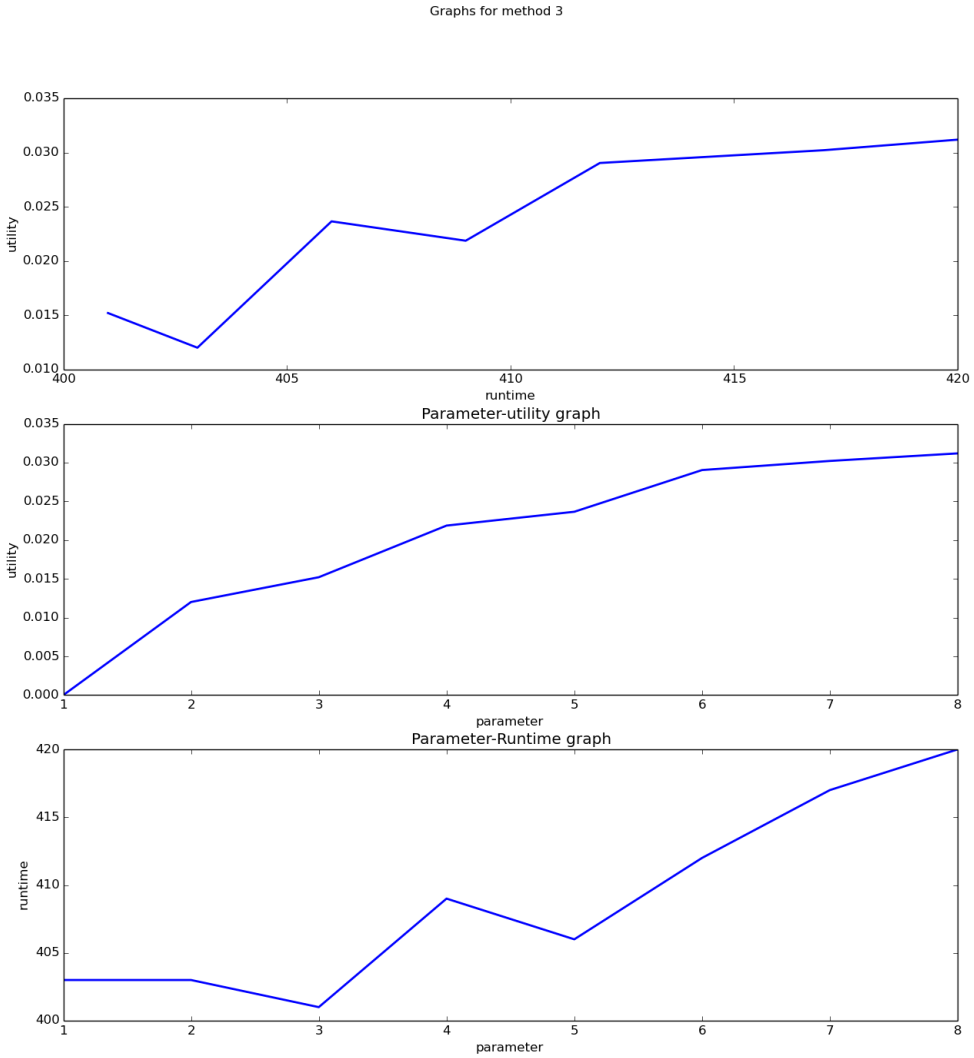

Figure 3.6: Trade-off analysis of method-3

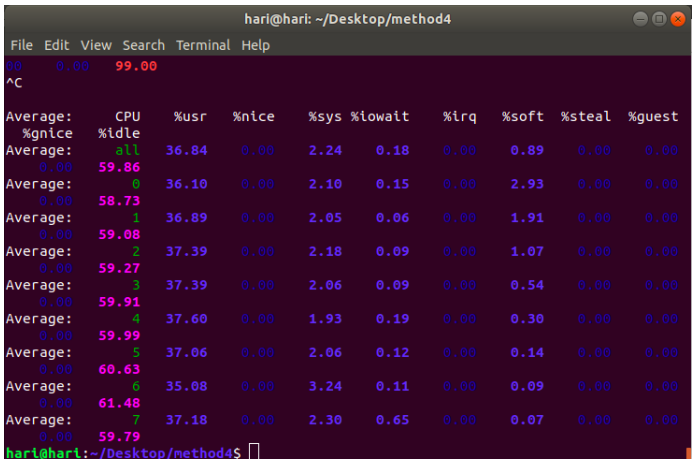

Figure 3.7: CPU usage for 4 threads

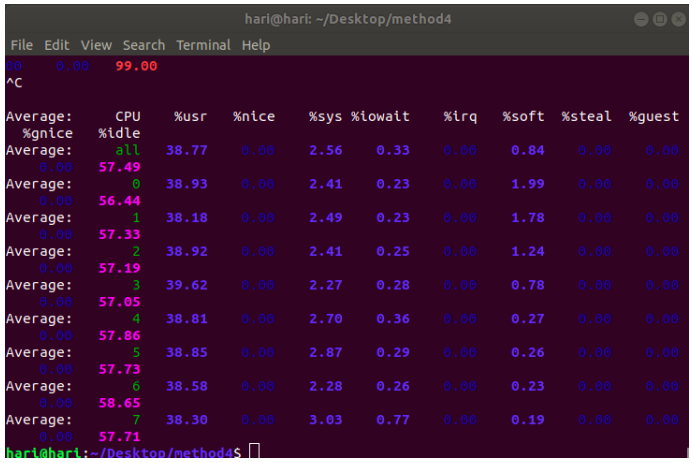

Figure 3.8: CPU usage for 5 threads

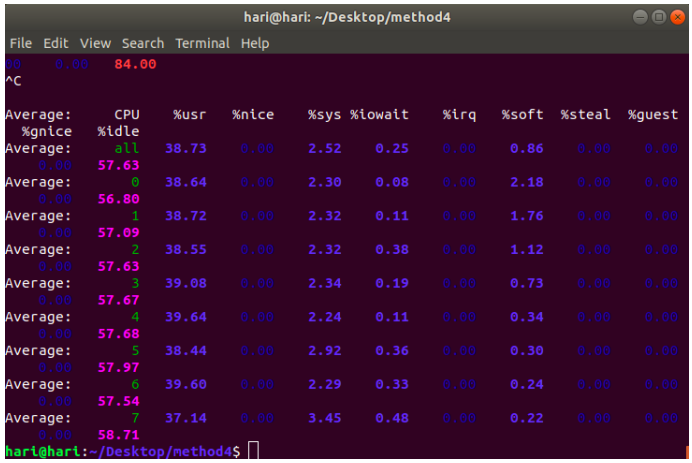

Figure 3.9: CPU usage for 6 threads

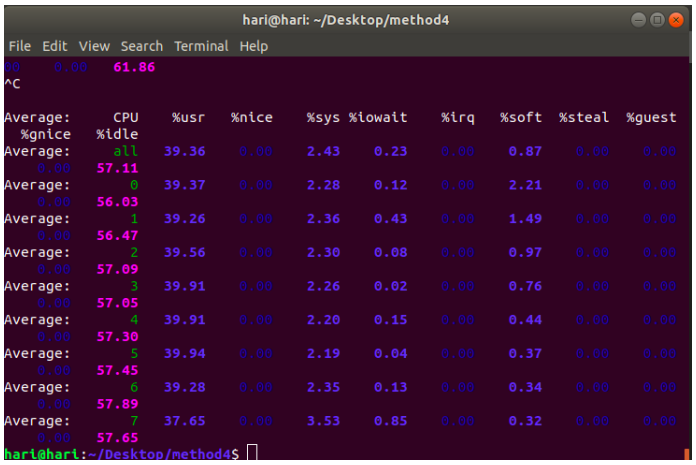

Figure 3.10: CPU usage for 7 threads

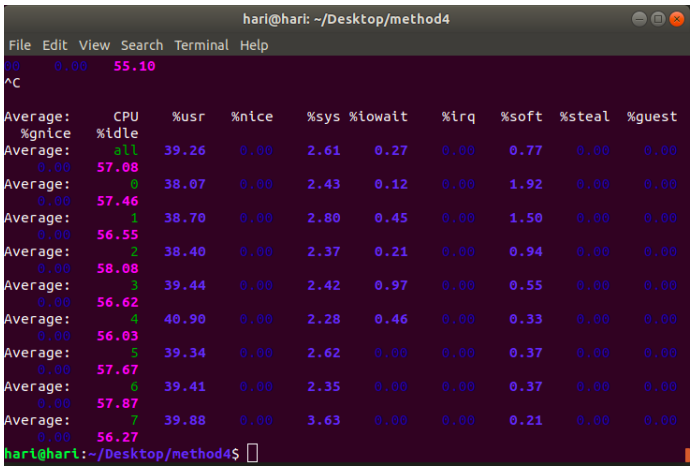

Figure 3.11: CPU usage for 8 threads

3.4 Method-4

From Figure 3.6 we can correctly infer that as the number of threads increases runtime decreases which is due the added parallelism that each thread brings allowing more frames to be computed simultaneously .

In Figure 3.7 we have plotted the queue density values for different number of threads, as the error is very less it seems like all the lines are overlapped.

From figure 3.8 we can see a trend that the total error increases as the number of threads increases this can be because many processes are happening in parallel.

It is to be noted that the error values are very small in comparison to the runtime benefit we get from running the program on multiple threads.

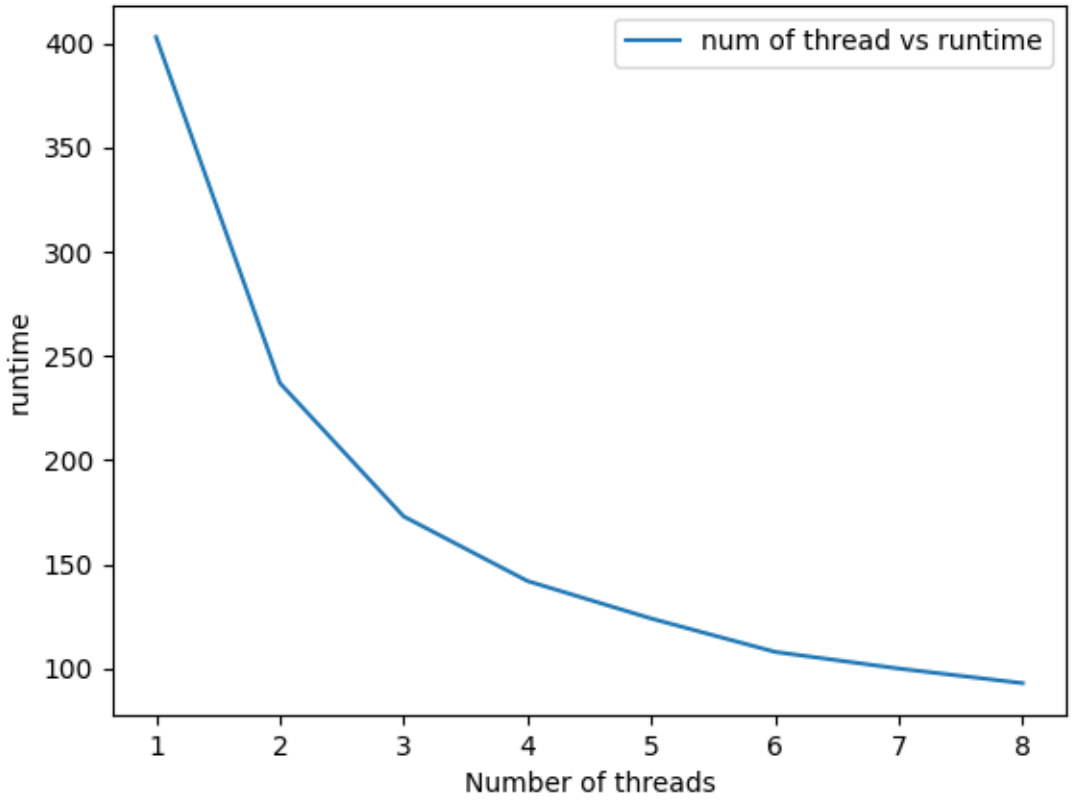

Figure 3.12: Number of Thread vs Runtime

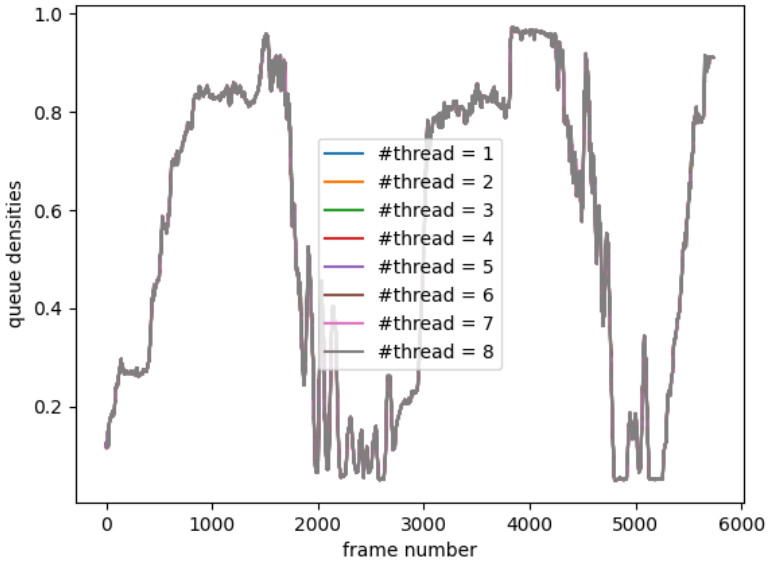

Figure 3.13: Queue density comparison for various threads

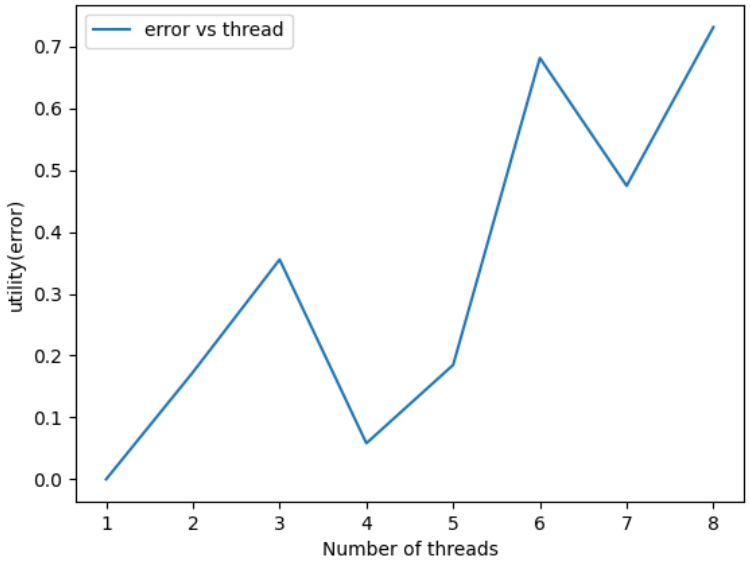

Figure 3.14: Number of threads vs Total error(utility)

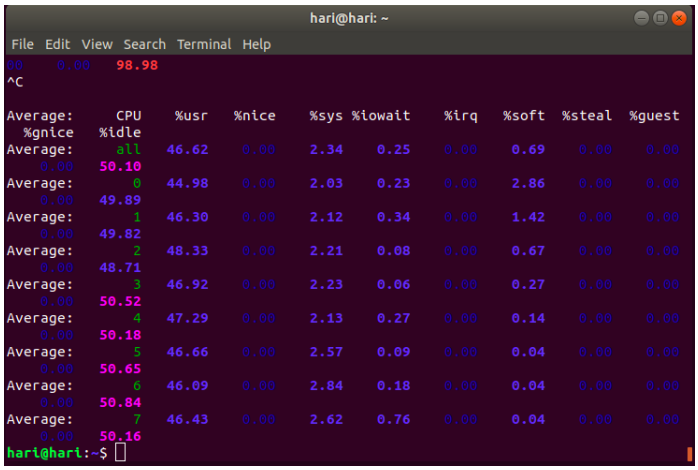

Figure 3.15: CPU usage for 2 threads

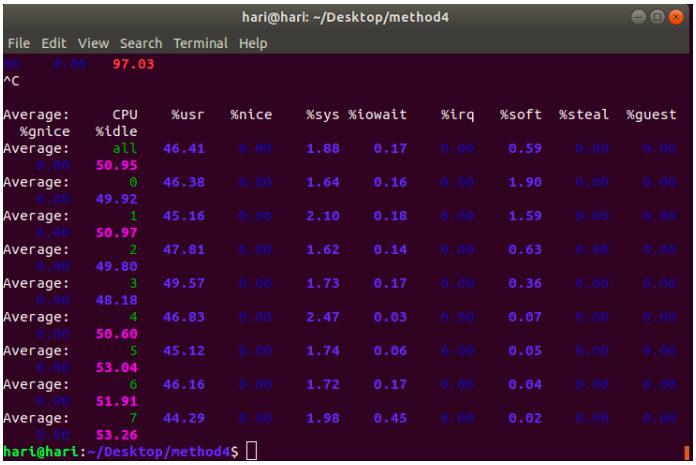

Figure 3.16: CPU usage for 3 threads

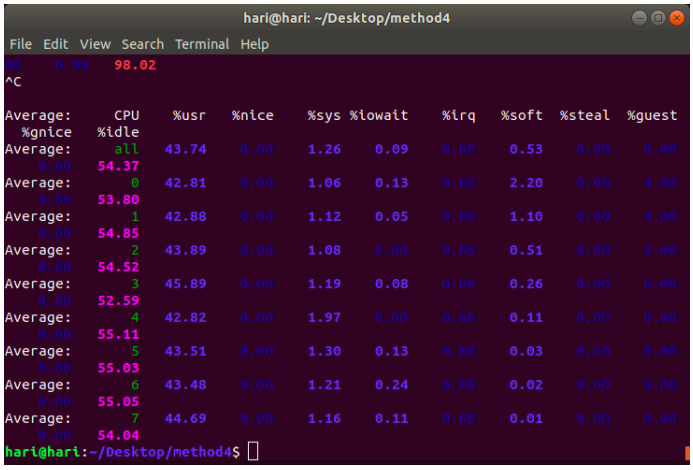

Figure 3.17: CPU usage for 5 threads

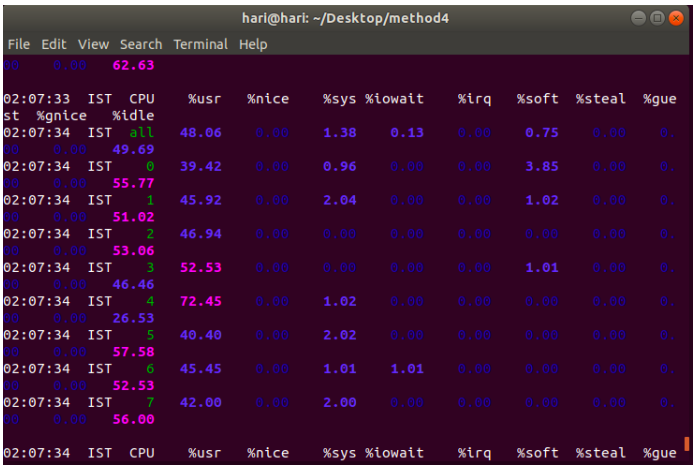

Figure 3.18: CPU usage for 6 threads

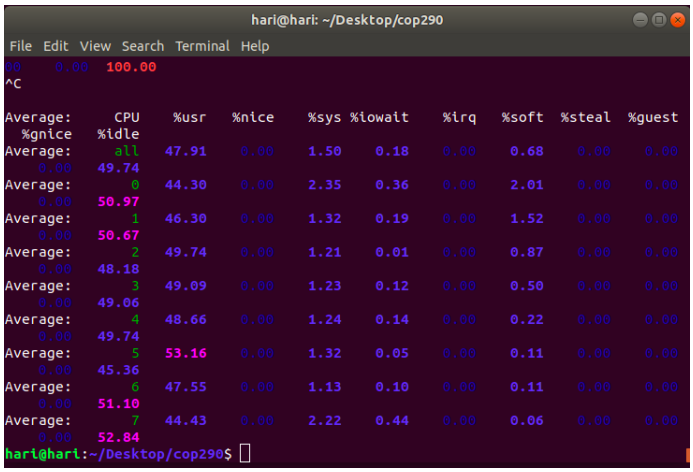

Figure 3.19: CPU usage for 7 threads

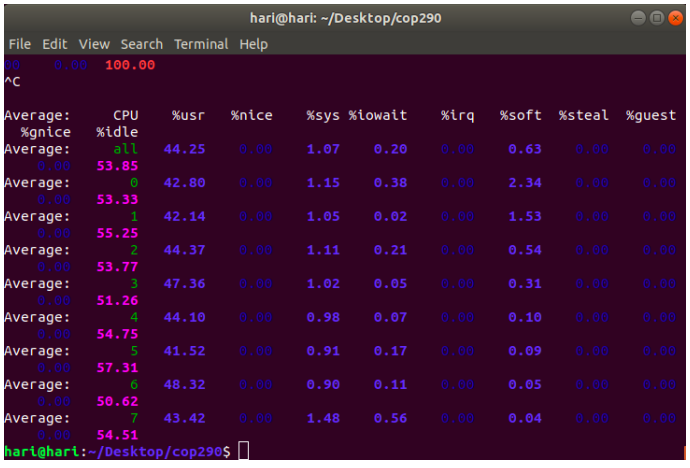

Figure 3.20: CPU usage for 8 threads

Conclusion and Future Work

4.1 Project Summary

We have successfully conducted the trade-off analysis on the four methods and drawn the inference that sub-sampling of frames should be used to decrease runtime and error ,the resolution can be decreased to a optimum level such that error doesn't increase beyond a threshold(according to user preference) while the runtime decreases ,multiple threads should be used to compute queue density as the runtime decreases significantly and the error increase is also negligible .

4.2 Areas of Improvement

We had done trade-off analysis mainly for the queue density as that was the intent of the project but we would like to analyse the dynamic density part as well and improve that part depending on the analysis result.

4.3 Future Works

We will try to test the trade-off for the dynamic density method by comparing sparse vs dense optical flow for analysis and work on improving the runtime of that method. Moreover our longer objective will be to create a traffic control system.