Understanding and Analyzing Trade-offs in Software Design

Aman Gupta & Arpit Chauhan

A report submitted for the course
COP290 Design Practices
Supervised by: Prof. Rijurekha Sen
Indian Institute of Technology Delhi

March 2021

© Aman Gupta & Arpit Chauhan 2021

DRAFT – 31 March 2021

Except where otherwise indicated, this report is our own original work.

Aman Gupta & Arpit Chauhan
31 March 2021

Abstract

When we build software, accuracy of the output or utility might not be the only metric to optimize. Sure, if we are estimating traffic density on road, we should output the correct density values. But maybe latency is also an important metric, that for a given input frame, we get output within a small time. Maybe throughput is an important metric, that every unit time, the code generates high number of outputs. Maybe keeping the processor temperature under control is necessary, if the software is deployed on a roadside embedded board, where there is no AC and ambient temperatures in Delhi shoots to 45 degree Celsius in summer. Maybe energy is an important metric, if the roadside embedded board is solar-powered, and can generate only a limited amount of energy. Maybe we want to protect the software from hackers, and therefore security is vital too.

These metrics might be at conflict with each other. E.g. for higher throughput, processors might run at a higher frequency, thereby draining more power and heating up more. Putting security checks against hackers might make the code run slower. This aspect of having multiple metrics to optimize, which conflict with each other, is called trade-off. Software design might need careful trade-off analyses.

In this Assignment , we have done a utility-runtime tradeoff analysis for Queue Density Estimation (using background subtraction). We have used several different Methods/Parameters which are basically Software Variants to analyze the Trade-off as compared to baseline.

Contents

1. Metrics

For analyzing any Software , we must first decide some metrics on the basis of which we compare different designs by varying the involved parameters .Commonly used metrics include accuracy of the output or utility, latency, throughput, temperature of CPU , CPU Utilization , security, Runtime and energy used .

1.1 Utility

Mean Squared Error of Queue Density from the baseline is an important metric considered in all the methods . The lower this error , higher is the utility of the particular design. It helps us to gauge the accuracy/correctness of the particular method as compared to the baseline method.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

1.2 CPU Utilization

Another important metric considered in Methods 3 and 4 is the **average CPU Utilization** during the program. It is a measure of the amount of Computing power being utilized by the program as compared to the maximum possible limit of the Machine/CPU.

1.3 Runtime

Runtime refers to the total execution time of the program on the given machine . It is a machine dependent metric hence all the trade-off analysis involving running time must be done on the same machine and in similar environment . To get the accurate runtime , it must be ensured that no other major process is running in the background .We have used the Chrono library of C++ to accurately measure the video processing time in different cases.

2. Methods

2.1 Method 1 - Sub-Sampling Frames

In this method , we processed every xth frame , i.e. processing frame N followed by frame N+x and use the value obtained for N for all the intermediate frames . Here ,we consider the Runtime and Utility metrics to analyse the trade-off . Utility is estimated by the inverse of Mean Squared Error calculated with x (Number of frames dropped+1) as the **parameter** and x=1 as the **baseline**.

2.2 Method 2 - Reducing Resolution

Here , we reduce the resolution for each frame before applying background subtraction . Since lower resolution frames are processed faster but might have errors as compared to baseline , Runtime and Utility are important metrics for which the trade-off has been analysed . Utility has been estimated by the inverse of Mean Squared Error. The **parameter** here is the factor by which the Resolution is reduced . **Baseline** is when this factor equals 1.

2.3 Method 3 - Spatial Threading

In this method , we trim the image into multiple parts and then assign each part to a separate thread for background subtraction . **Parameter** is the number of splits which is also equal to the number of threads . CPU Utilization , Utility and Runtime metrics have been considered in this method for trade-off analysis. **Baseline** is when there is a single thread .

2.4 Method 4 - Temporal Threading

In this case we split the work temporally across threads(application level pthreads) by assigning consecutive frames to different threads for processing . **Parameter** here is the number of threads with **baseline** taken as the single thread case. CPU Utilization is an important metric here.Utility and Runtime metrics have also been considered in this method for trade-off analysis.

2.5 Sparse vs Dense Optical Flow

Optical flow refers to the the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. In the context of this assignment traffic at the road is the moving object and the camera is the observer. Optical flow is further of two types namely, Sparse and Dense Optical flow. In sparse optical flow we track we good features in the images, while in dense optical flow we track all the moving pixels. We try to compare the approaches to estimate which one is better for the prediction of traffic density on road. Dense optical flow is taken as the **baseline** and the **utility** is the inverse of the Mean squared error from the Dynamic density of baseline.

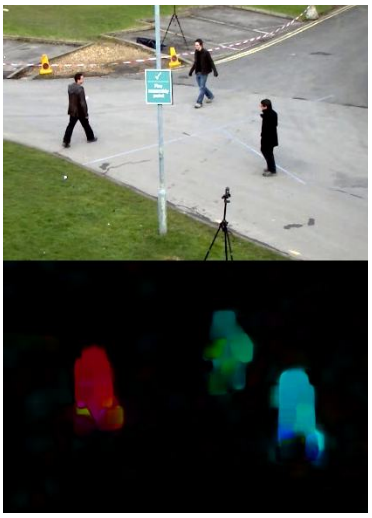


Figure 2.1: Dense Optical Flow

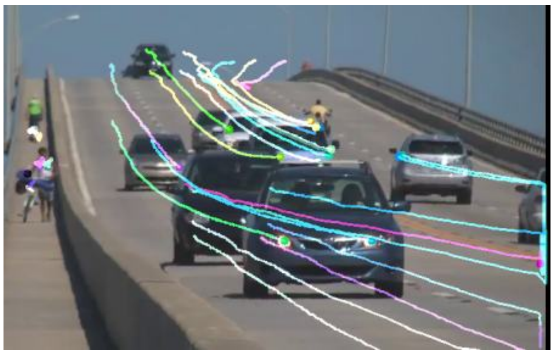
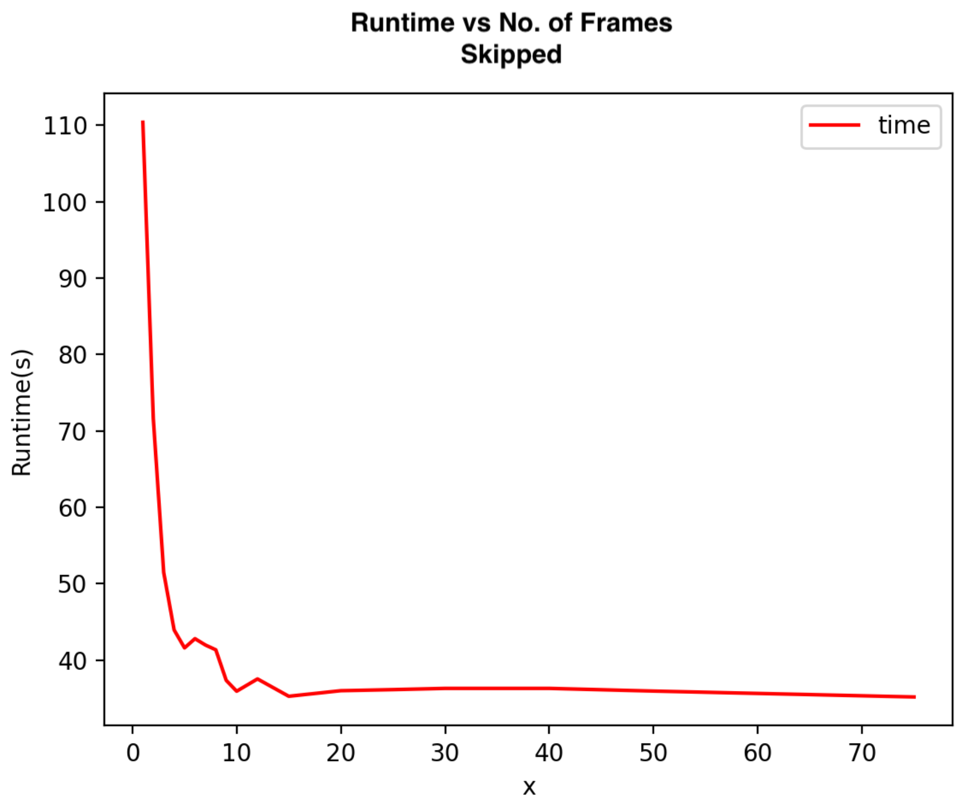
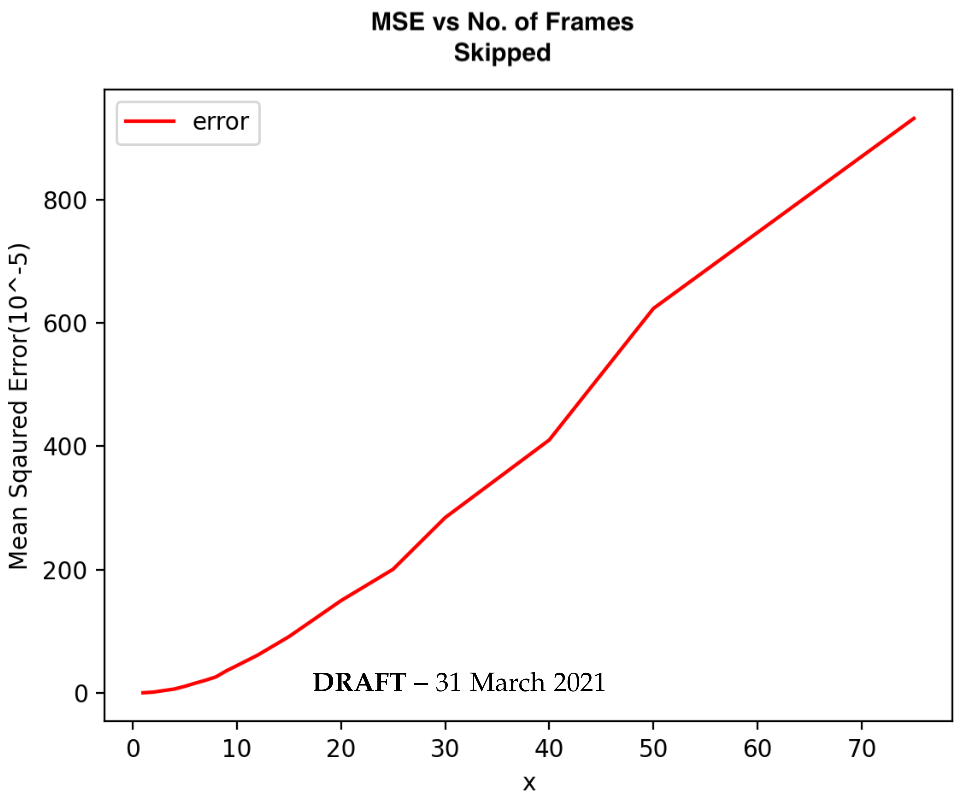


Figure 2.2: Sparse Optical Flow

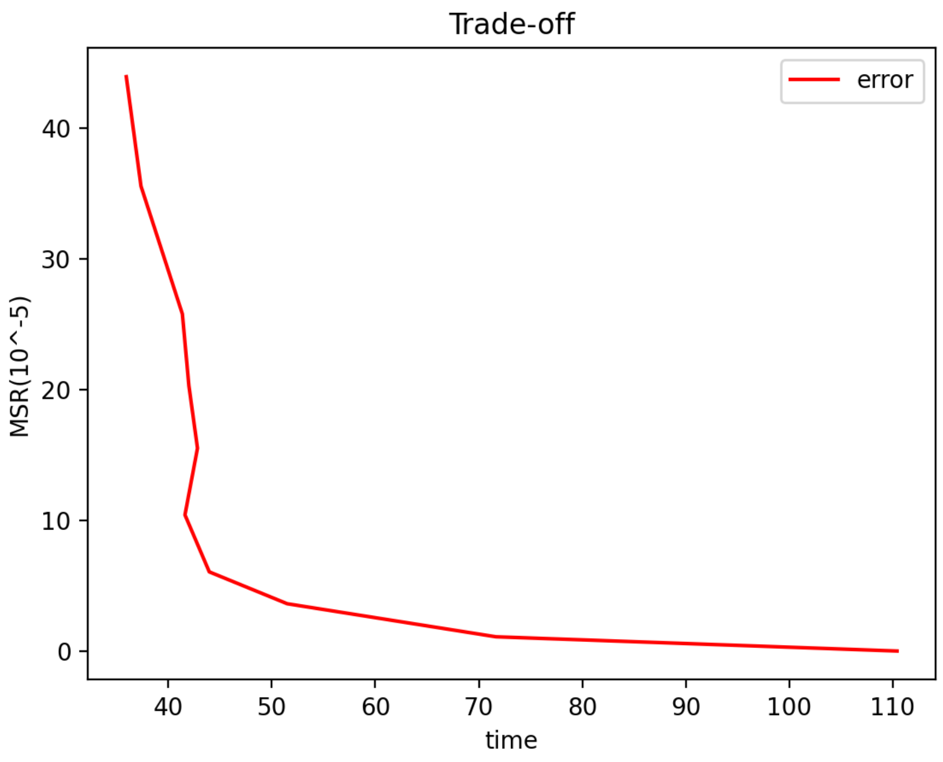
3. Trade-off Analysis

3.1 Sub-Sampling Frames



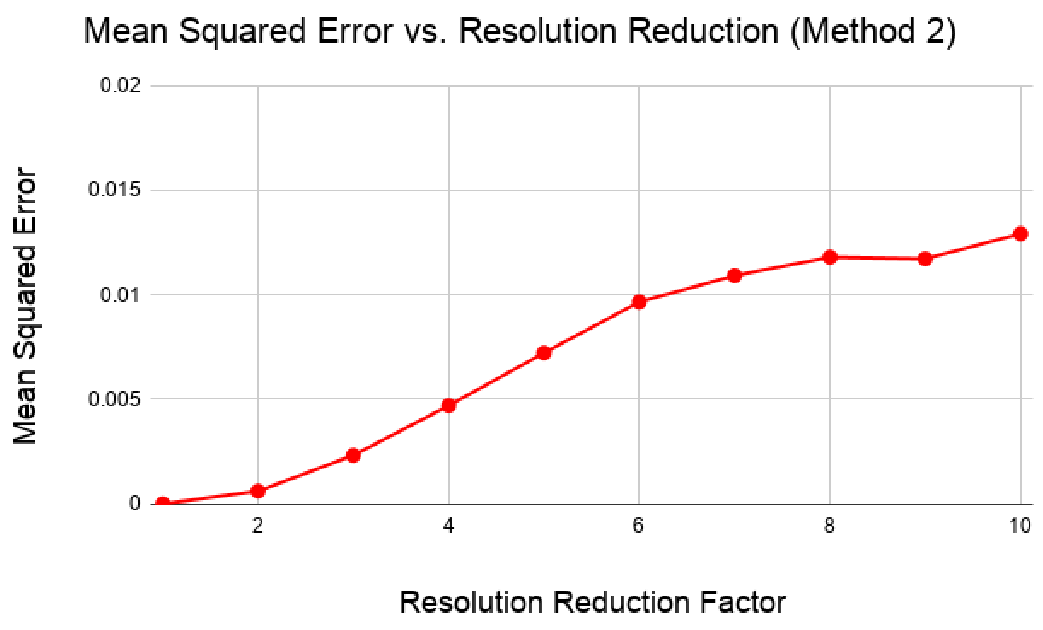


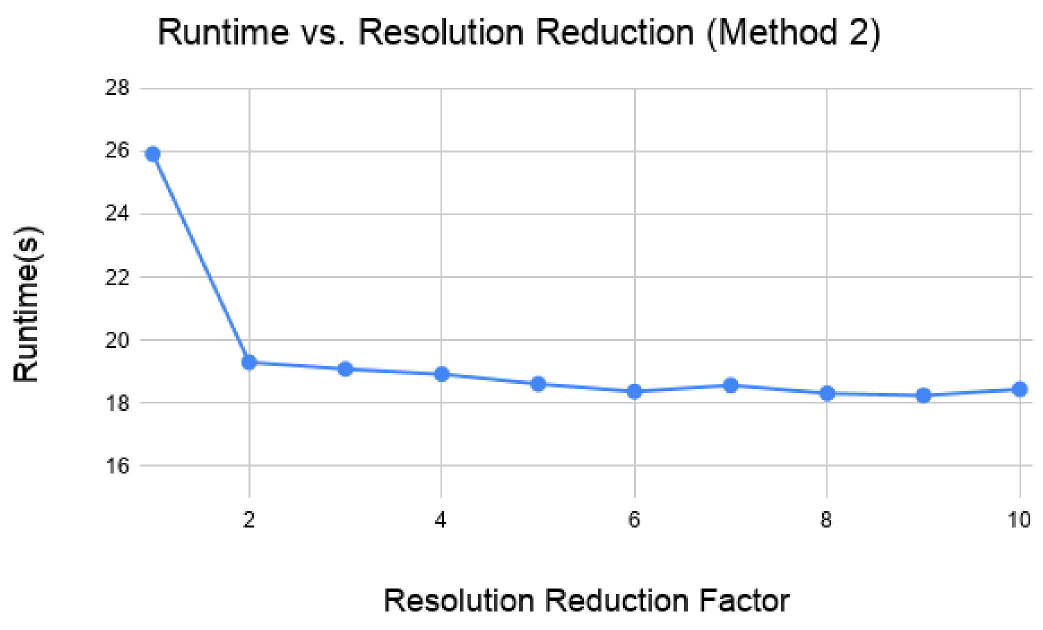
DRAFT – 31 March 2021

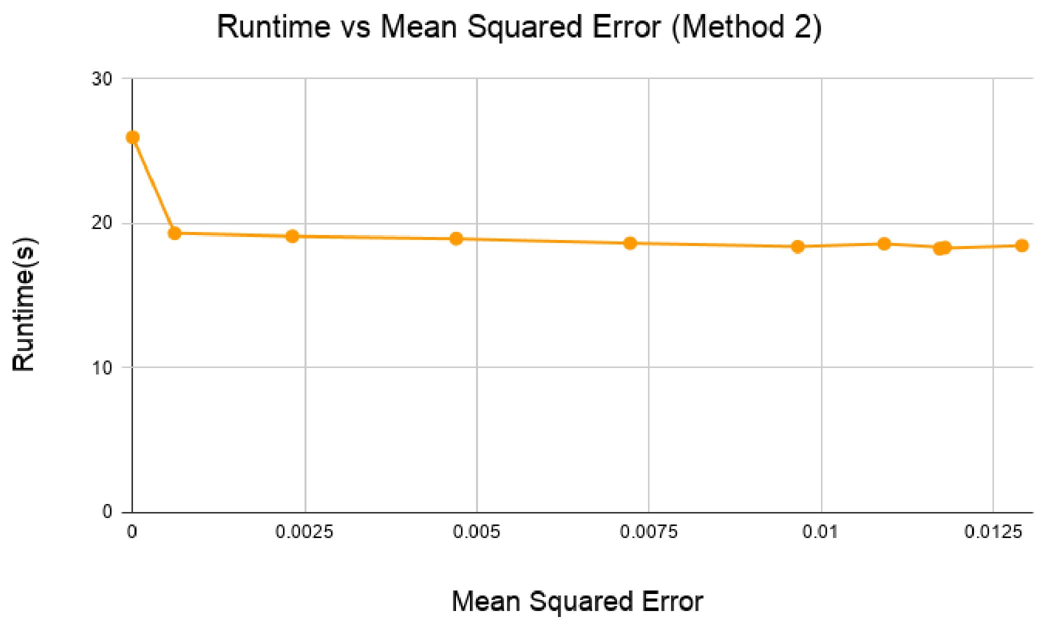


EXPLANATION
Since we are dropping x-1 frames every time we process a frame , the total number of frames to be processed reduces by a factor of x , hence the total runtime of the program is bound to decrease as x increases, thus explaining the 1st graph.As we keep increasing x,the runtime approaches a contant value which corresponds to other operations in the program which are independent of FPS being processed.
Since we are using the value obtained from processing frame N for all the frames between N and N+x , error is induced which is bound to increase as we increase x,thus explaining the 2nd plot.
As we increase x , the runtime of the program decreases which is a good thing but the error as compared to baseline also increases , thereby reducing the utility of the software. Hence , there is a Utility-Runtime tradeoff which explains the 3rd plot.

3.2 Reducing Resolution







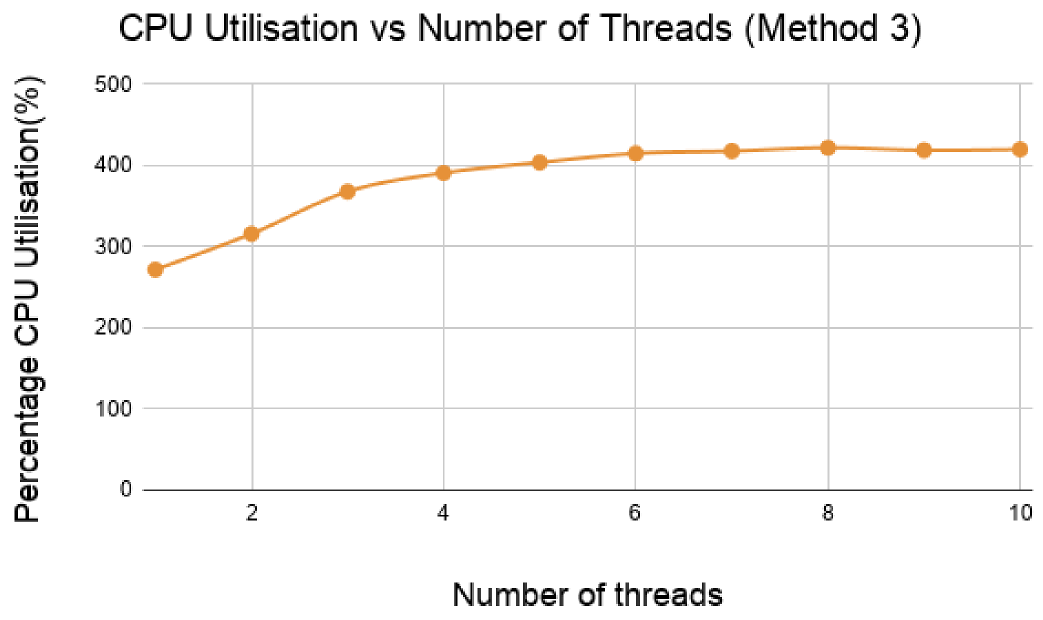
<u>EXPLANATION</u>:
In this method we compare the performance of the background subtraction algorithm by changing the resolution of the image to be processed. While changing the resolution we ensure that the **aspect ratio** of the image remains **same**. So that the width and breath of the input image changes by the same factor. We have taken this factor to be the **parameter Resolution reduction**, used for comparison of the performance of background subtraction.

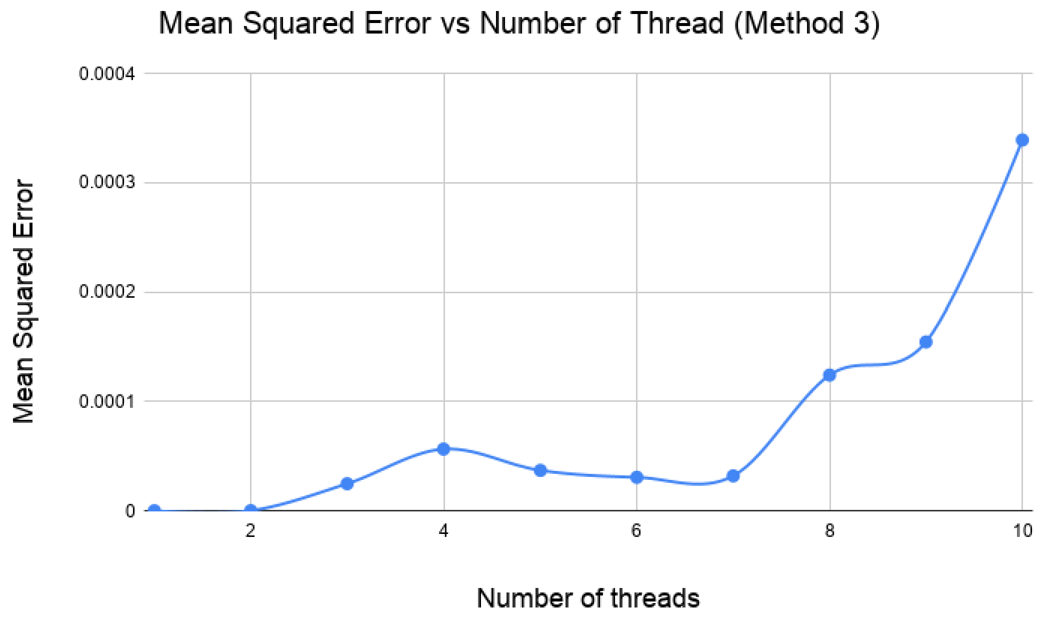
As apparent in the figure one that the Mean Squared Error increases on increasing the the resolution reduction factor. This occurs because as we increase this factor reduces the size of the image and many pixels are grouped together to form an aggregate. In the baseline algorithm each of these pixels had individual effect on the density, but now these aggregates causes change in queue density. This change is the main source of error in this method. As we increase the resolution more number of pixels are aggregated hence causes larger error.

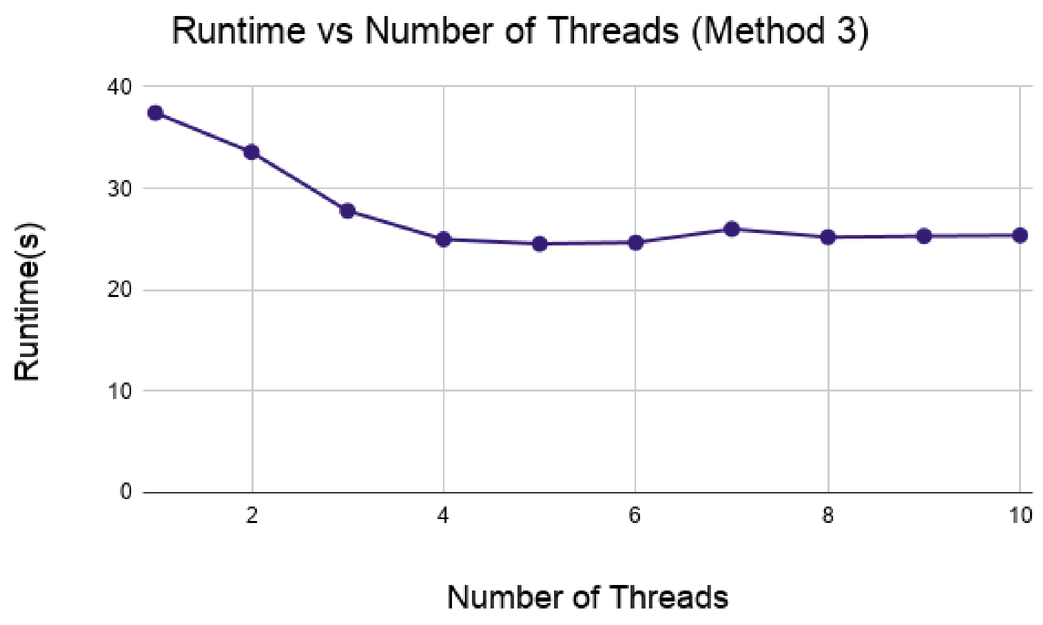
Also since on increasing the resolution reduction factor the size of image decreases. Hence the time taken to process them also decreases. However as we increase the factor the time taken to generate the resolved image also increases. This acts as an **overhead** on the baseline algorithm. Therefore, the graph obtained is combined effect of both these reasons.

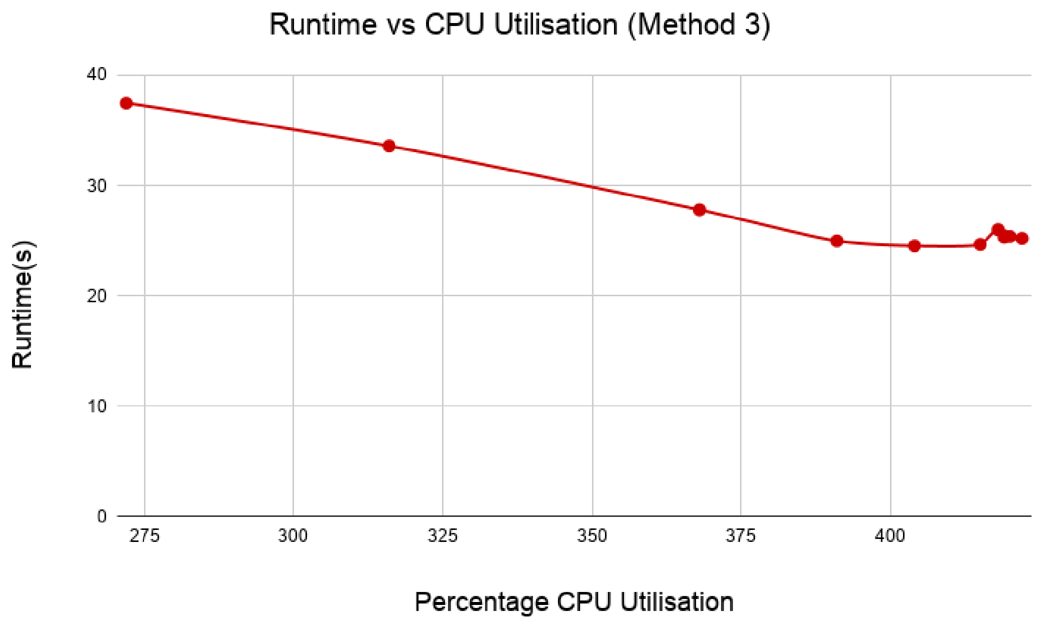
Plot 3 shows the **trade-off** between the Runtime and the utility on varying the parameter, Resolution reduction. We observe the general trend that the Mean Squared Error increases with decreasing the Runtime, that is **faster process increases error** in the output.

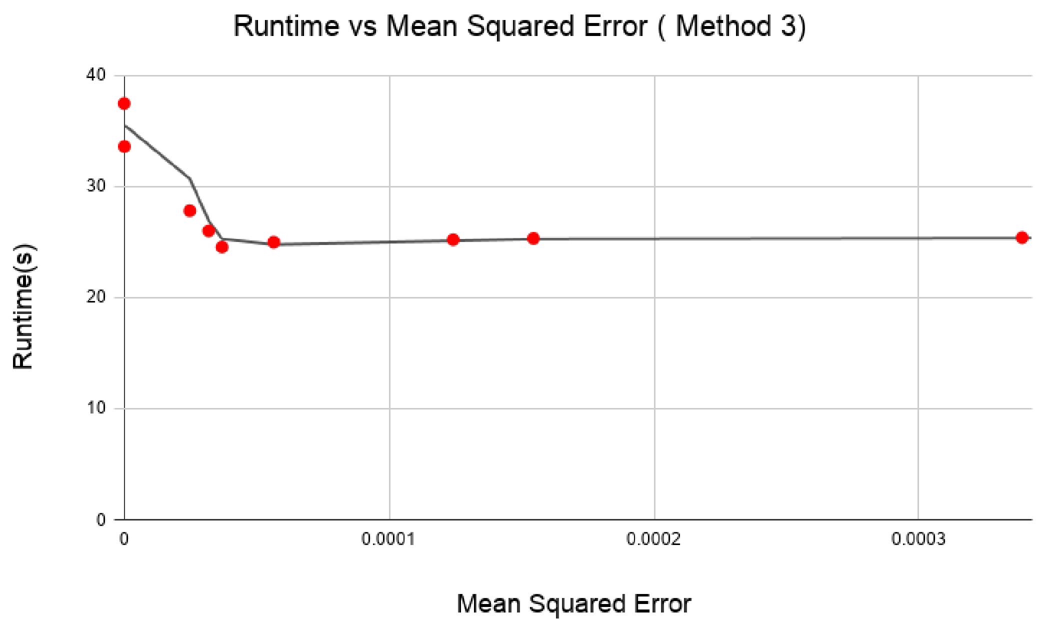
3.3 Spatial Threading





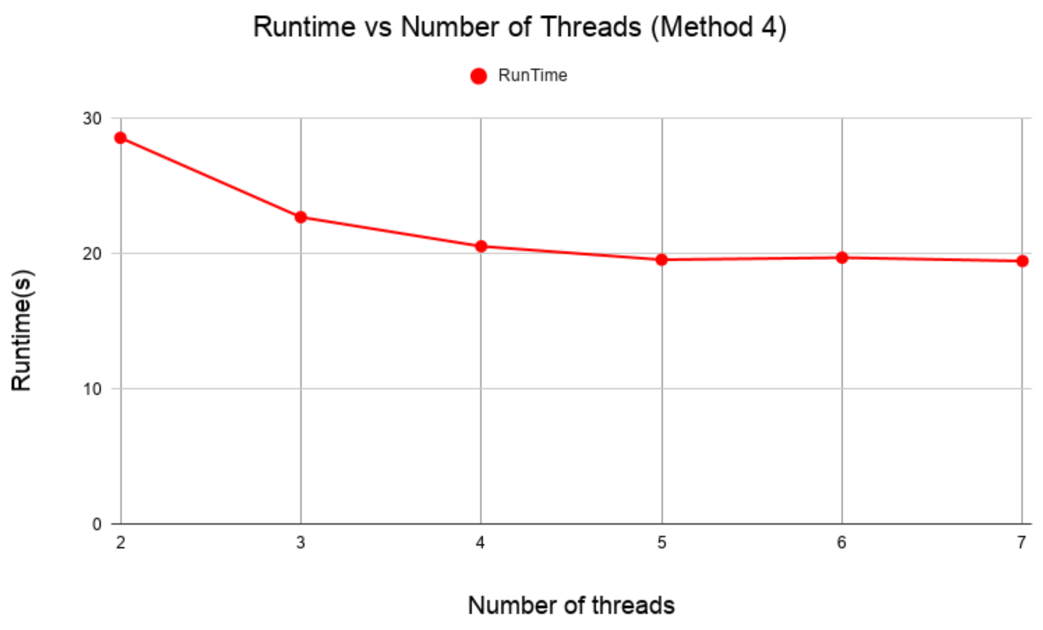

Runtime vs Number of Threads (Method 3)
Runtime(s)
Number of Threads
0
10
20
30
40
2
4
6
8
10

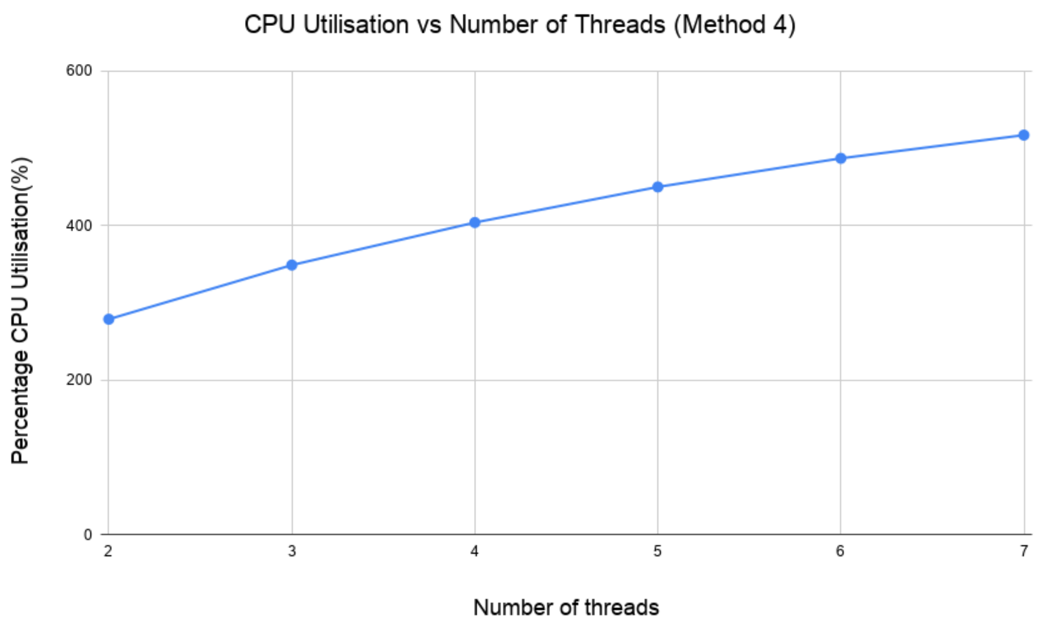

Runtime vs CPU Utilisation (Method 3)
Runtime(s)
Percentage CPU Utilisation
0
10
20
30
40
275
300
325
350
375
400

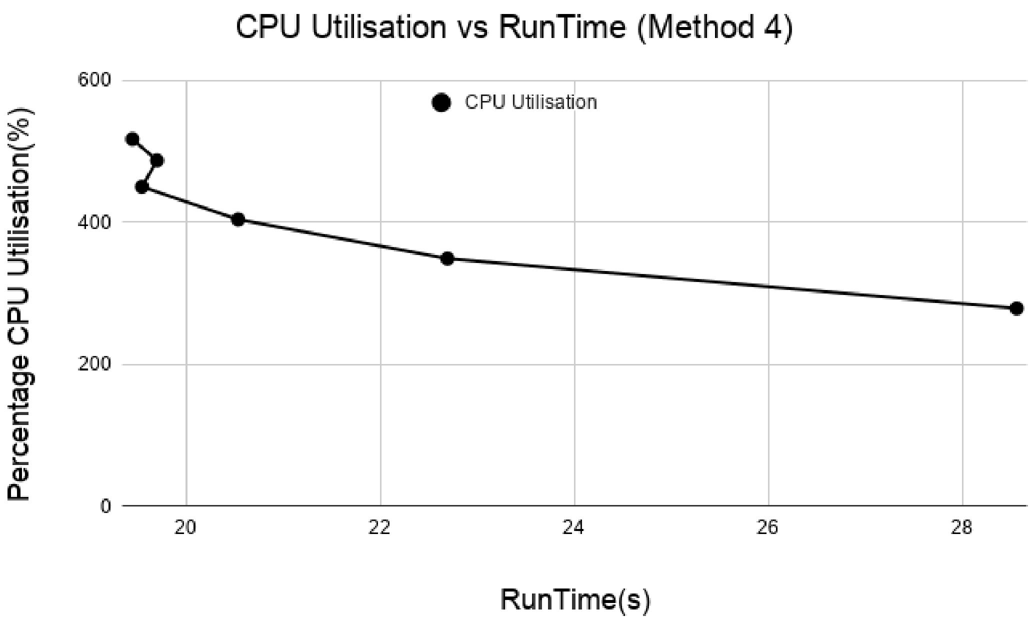


EXPLANATION:
In this method , we split each frame into multiple parts and then split the work spatially across application level pthreads .Since , the work to be done is now more spread among the cores of the CPU as compared to the baseline ,average CPU utilization of the program should increase by multithreading which explains the 1st plot. As we keep increasing the number of threads , average CPU Utilization starts saturating because all the cores get occupied and there is no further scope of improvement.
As we split the image into multiple parts , more number of edges are introduced which results in deviation of values from the baseline thus increasing the error and decreasing the utility as we increase the number of threads ,thus explaining the 2nd Plot. But this error is very small , hence it is safe to say that Utility is not affected much by multithreading.
Increase in CPU utilization means more work is being performed by the CPU at the same time thereby reducing the runtime of the program .So , when we increase the number of threads , increase in CPU utilization decreases the overall runtime of the program , thus explaining the plots 3 and 4.
Although error is not significant in this method , there is still a Utility-Runtime trade-off. Increasing threading reduces the Runtime but also increases the error , thereby reducing the utility , hence the trade-off which explains the plot 5.

3.4 Temporal Threading







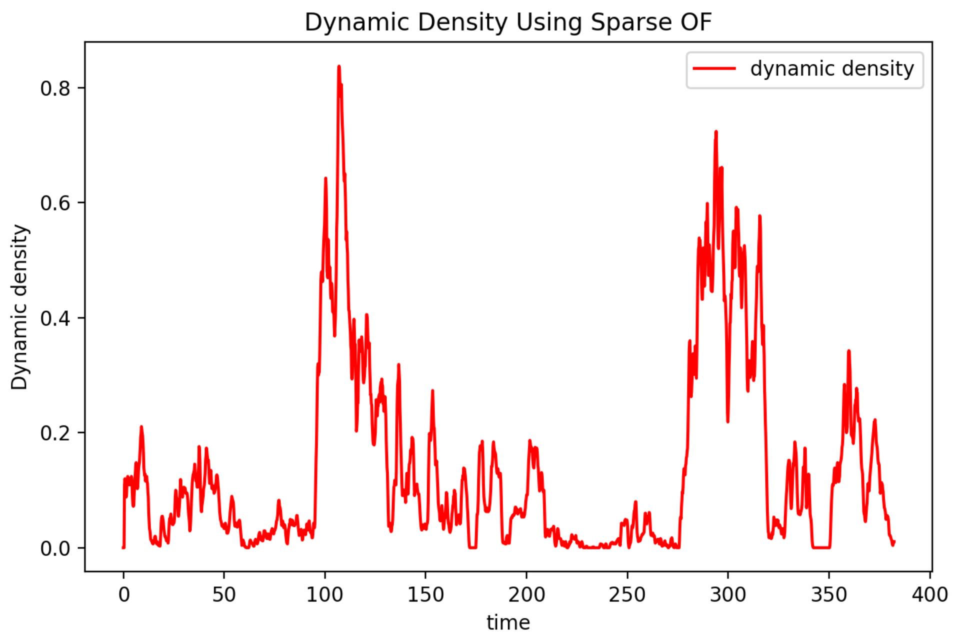
<u>EXPLANATION</u>:
In temporal threading consecutive frames of the video has been assigned to different threads. This causes **temporal parallelization** of otherwise sequential process. However this process is not completely parallel because only a number, equal to total number of threads, frames can be processed at a time. And as batch this process occurs sequentially with the next batch. Individual threads do both perspective correction and background subtraction of the frame assigned to the thread.

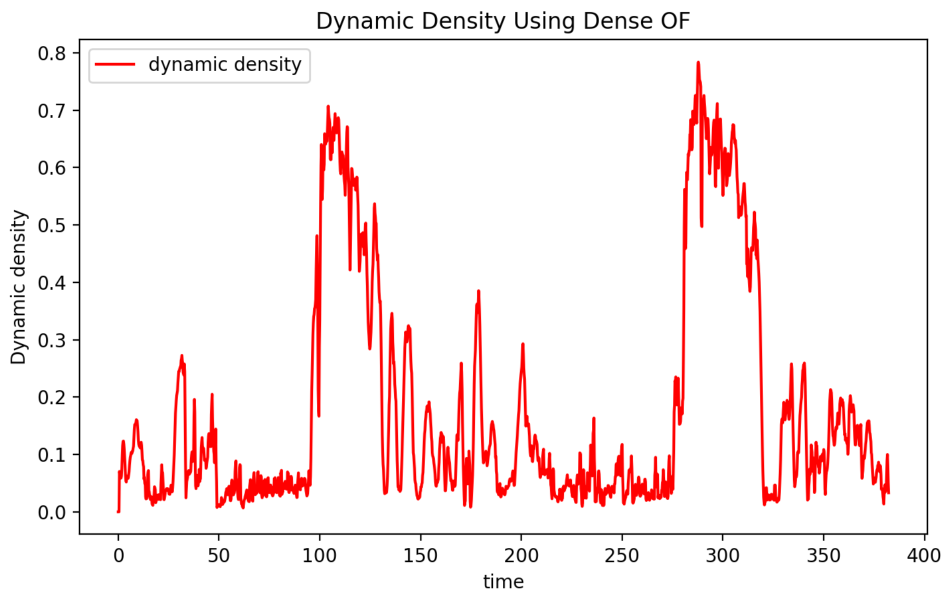
As we increase the number of threads the amount of concurrent processes increases as a result of which the number of frames that can be processed simultaneously, also increases. Hence the total run time decreases with an increase in the number of threads. The same pattern is apparent in the curve obtained. Though the curve becomes more or less constant after 5-6 number of threads because the CPU utilization is maximized and computation cannot be done any faster.

As the number of threads increases the percentage of CPU Utilization also increases as the machine tries to assign different threads to different CPU cores. This increment continues until the utilization reaches its saturation. After reaching saturation any further increase in the number of threads will not increase CPU utilization. A very similar trend is obtained in the graph.

Plot 3 displays the trade off between utility and Runtime, as apparent from the graph that the **Runtime decreases with increase in CPU utilisation**. This is because as the CPU utilization increases the computing power increases and hence the video is processed faster.

3.5 Sparse vs Dense Optical Flow for Dynamic Density





Mean Squared Error = 0.0092

Time taken by Dense Optical Flow = 70.46s
Time taken by Sparse Optical Flow = 13.40s

EXPLANATION:

As apparent from the time values sparse optical flow method is much faster but has some errors relative to the dense optical flow. Sparse optical flow just keeps track of some good features to track while dense optical flow checks all the moving pixels. Sparse optical flow gives you the flow vectors of some "interesting features" within the image. Dense optical flow attempts to give you the flow all over the image - up to a flow vector per pixel. Sparse also requires more amount of CPU utilisation as compared to dense optical flow. For this assignment Dense optical flow would be a better algorithm to predict the density of traffic on the road. This is because it has less error and also requires less amount of CPU Utilisation thereby causing lesser amount of heat. Moreover sparse optical flow is more noisy than dense optical flow.

4. Conclusion

- The number of thread increases the amount of CPU utilisation as it causes different parts of code to execute in different cores thus causing parallelisation of the sequential frame processes.
- The mean squared error usually increases with a decrease in the runtime. That is faster processes usually have higher error. Optimisation between runtime and utility is one of the most important design decision in any software development task.
- We noticed that the runtime is lesser in case of temporal threading as compared to spatial threading. This is because in the spatial threading extra time is required to split the frames into different frames. Also the error in case of temporal splitting is much less as compared to in case of spatial splitting. Hence temporal threading is much better than spatial threading in this case.
- Parallel computing saves time, allowing the execution of applications in a shorter wall-clock time. But at the same time it also increases the power consumption as it demands multi-core architectures.
- Reducing the resolution and the number of frames processed increases the run time but also induces some error in the outputs.