Rajdeep Singh Dhingra
Mrunmayi Bhalerao
2019CS10388
2019CS50425
cs1190388@iitd.ac.in,
cs5190425@iitd.ac.in

COP290
Task 1: Traffic density estimation using OpenCV functions

2021-03-31

Introduction

In this Assignment, We have first written a program which takes in a video of traffic and outputs the queue density ie. the number of cars present on the traffic light with flow in time.

We have then created various methods which use various parameters to test out which method of software design is both accurate and fast.

1 Metrics

In this assignment we will vary parameters and methods to compare the metrics of various approaches to achieve the same task. This is very necessary in software design.

Over here we will have mainly two ways to check how good a program is -

1. **Utility/Accuracy -** If the output of a program is as close to the actual/expected value then the program is more accurate. That means the output of the program is more useful and therefore has higher utility.
2. **Run Time/Speed -** Better programs are those which run in less time. Having higher speed would imply faster results. This is one of the main factors of a good software. A good software must run as fast as possible.

Although other factors include memory usage, CPU usage and energy consumption. Dealing with all of them at the same time will require some more sophisticated methods. However, we will touch upon these parameters in the methods which use multi-threading.

1.1 Utility Measure

We define our utility measure as the error/offset from the baseline data. We compute the Root Mean Squared Error from the baseline data.

$$\text{Error} = \sqrt{\frac{\sum_{\text{over all Frames} F} (d[F] - db[F])^2}{\text{num of frames}}}$$

where $d[F]$ is the calculate queue density in Frame F and $db[F]$ is the baseline queue density in Frame F.

Since Accuracy is inversely proportional to Error, We define

$$\text{Utility Measure}(U) = \frac{1}{\text{Error}}$$

1.2 Runtime Measure

We will calculate Run-time measure using the "Chrono high precision clock" provided in the chrono C++ Library.

chrono::high_precsion_clock

We define the runtime to be the time it takes from start of reading the video to end of processing each frame. This runtime is averaged over many values to remove random variations from it.

2 Methods

We have implemented 4 methods apart from the main method which we used to compute the baseline. The methods are as follows:

- Method producing the Baseline - We have used background Subtraction to produce the queue density. Firstly, we crop and perform a perspective transform. Then apply background subtraction from an empty traffic to get a black white image.

- Method 1: Sub-Sampling Frames - We have skipped frames to decrease the processing time. If we process the frame N then we next process the frame N+X only.

- Method 2: Reduce resolution for each frame - Each frame has a reduced resolution which decrease the processing time per frame.

- Method 3: Split work spatially across threads - Each image is divided into various sub-parts and each part is processed independently. This might lead to loss in border data. This speeds up the process as each thread has to act on fewer pixels.

- Method 4: Temporal splitting of work across Threads - Each frame is passed to an independent thread, so that multiple frames can be processed concurrently. This speeds up the process and each thread need to process a fewer number of frames.

Method No	Method Name	Parameters
0	Baseline(original)	VideoFile
1	Sub-Sampling Frames	VideoFile, SkipRate
2	Reduce Resolution	VideoFile, Width, Height
3	Spactial Spilting	VideoFile Threads
4	Temporal Spilting	VideoFile Threads

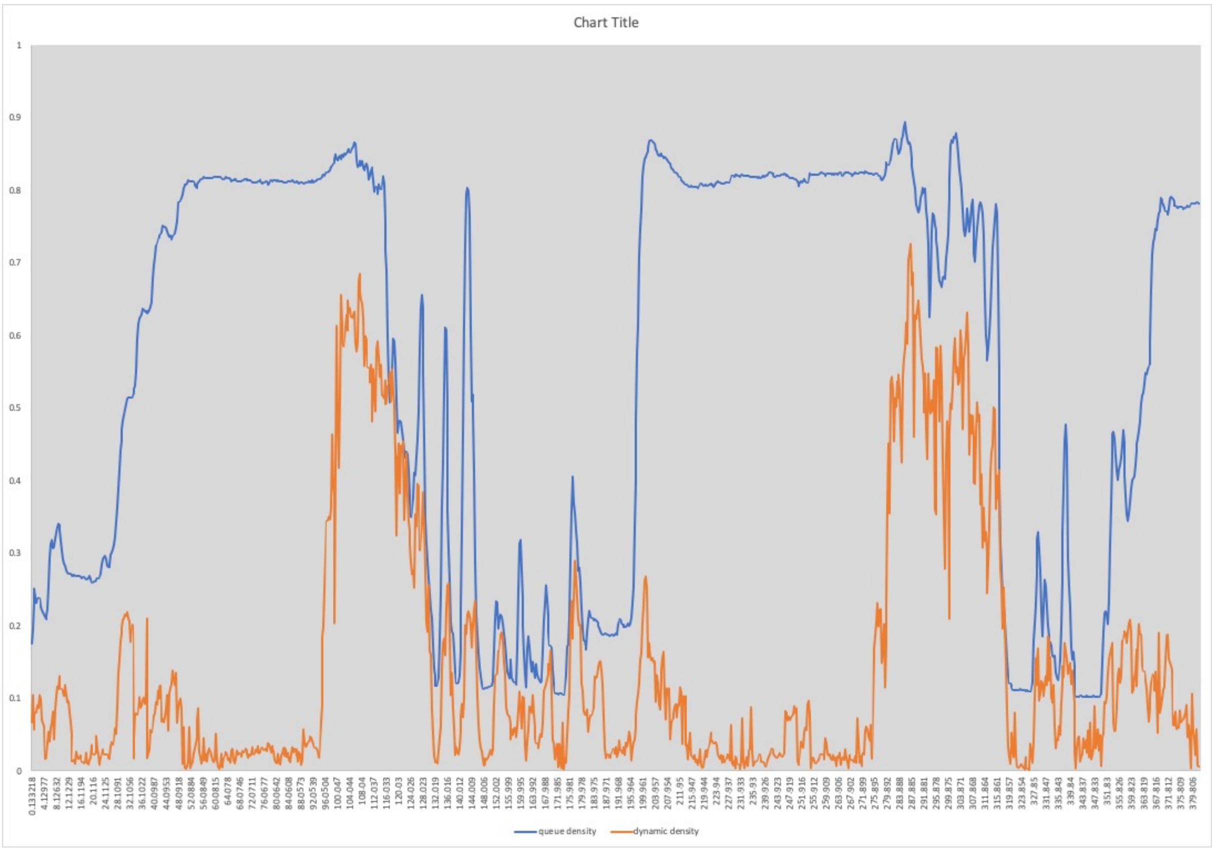


Figure 1: Queue Density and Dynamic Queue Density (Baseline)

3 Trade-Off Analysis

We have analysed various methods and varied their parameters. We plotted graphs of Run-time vs Utility, Run-time and Utility vs Parameter and Memory, CPU usage vs Parameter.

3.1 Method 1

Here we skip many frames. This causes loss of data but gives less information to process. Generally not much changes between very nearby frames and this fact is exploited by this method.

Parameter	Time	Error
1	40.265	0
2	26.2	0.004319
3	21.91	0.00682289
4	19.11	0.00987614
5	19.04	0.0121187
6	17.44	0.0146212
8	16.91	0.0199776
9	16.49	0.0220639

Figure 2: Table

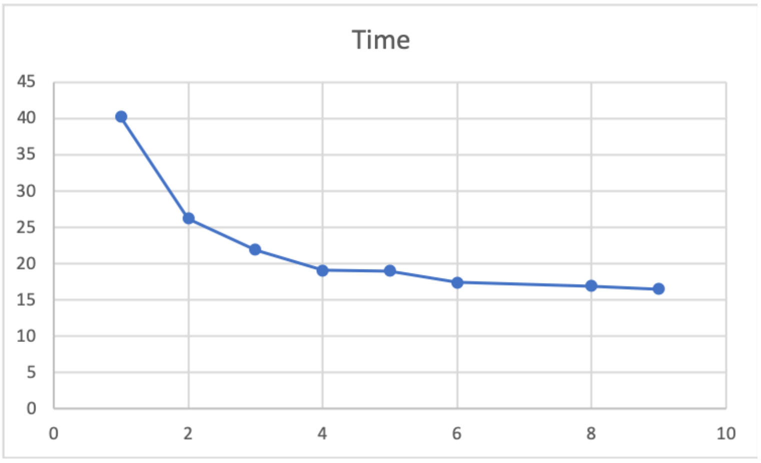


Figure 3: Time vs Parameter

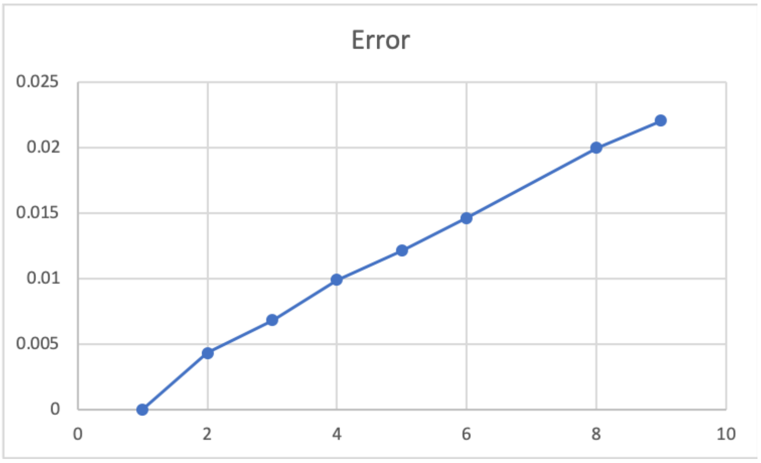


Figure 4: Error vs Parameter

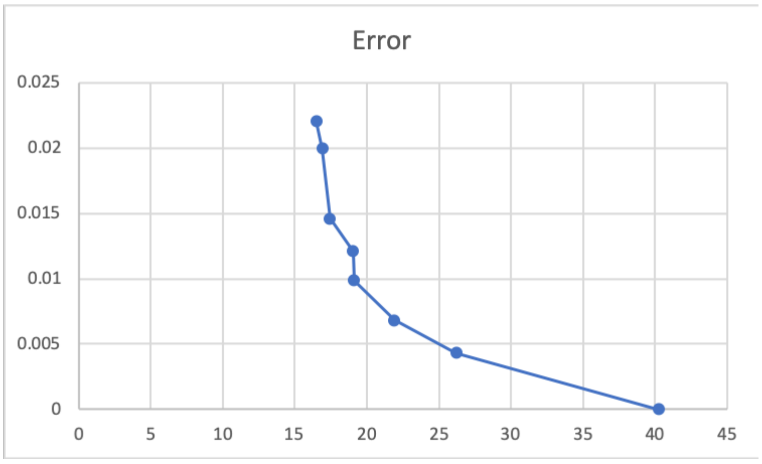
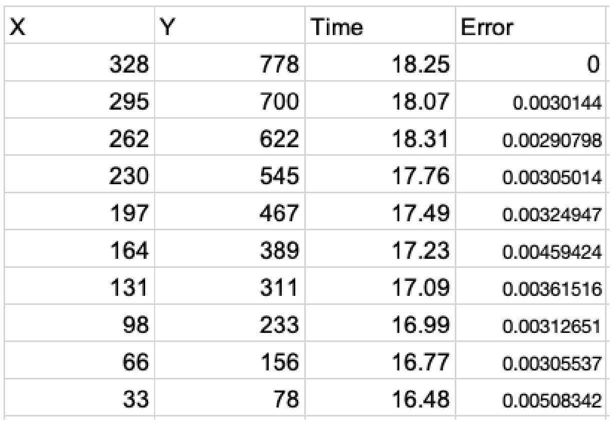


Figure 5: Error vs Runtime

Increasing skip results in the processing of a smaller number of frames and thus reduces runtime. The runtime follows an almost inverse trend with the skip factor. The error will increase although.

3.2 Method 2

Here we reduce the resolution of each frame and thereby reducing the processing time for each frame.



X	Y	Time	Error
328	778	18.25	0
295	700	18.07	0.0030144
262	622	18.31	0.00290798
230	545	17.76	0.00305014
197	467	17.49	0.00324947
164	389	17.23	0.00459424
131	311	17.09	0.00361516
98	233	16.99	0.00312651
66	156	16.77	0.00305537
33	78	16.48	0.00508342

Figure 6: Table

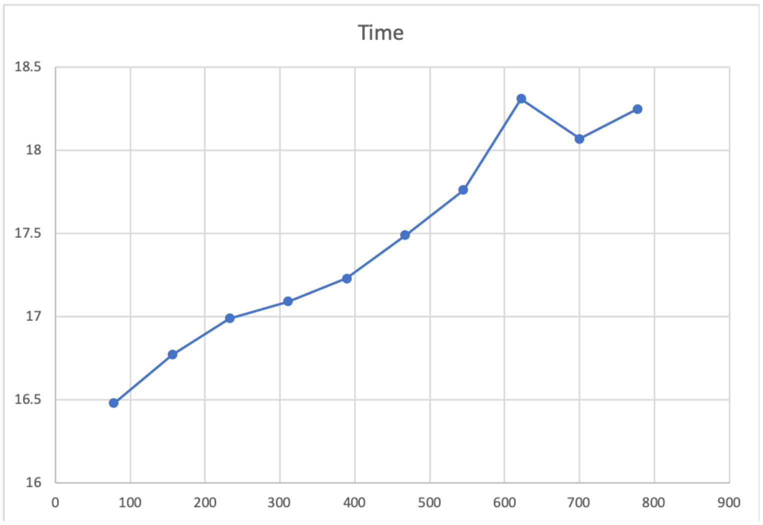


Figure 7: Time vs Parameter

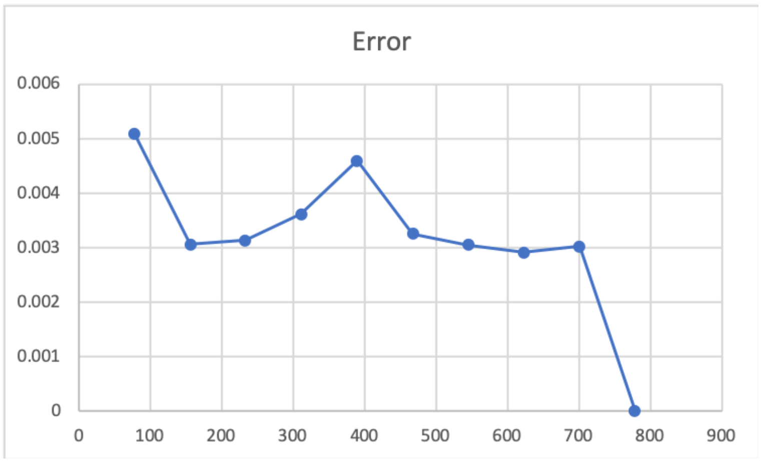


Figure 8: Error vs Parameter

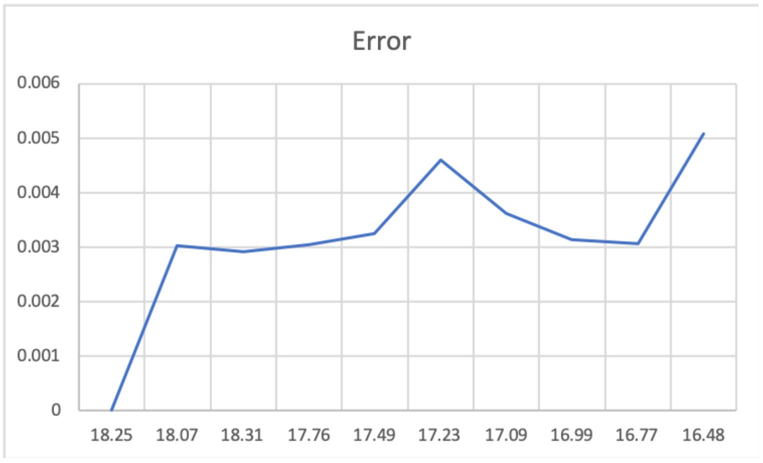
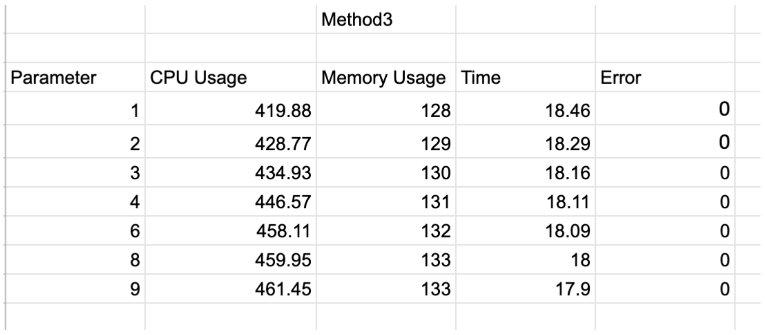


Figure 9: Utility vs Runtime

As we decrease the size of the image. There will also be time taken for processing of the image to reduce it's size. This will cause less gains in run-time as we decrease the size of the image. The errors will also increase significantly as the size of the image reduces.

3.3 Method 3

Here the work is spilt spatially. Each image is divided into various parts and each part is processed by the thread. This makes each thread process lesser amount of pixels and hence increases execution time.



		Method3		
Parameter	CPU Usage	Memory Usage	Time	Error
1	419.88	128	18.46	0
2	428.77	129	18.29	0
3	434.93	130	18.16	0
4	446.57	131	18.11	0
6	458.11	132	18.09	0
8	459.95	133	18	0
9	461.45	133	17.9	0

Figure 10: Table

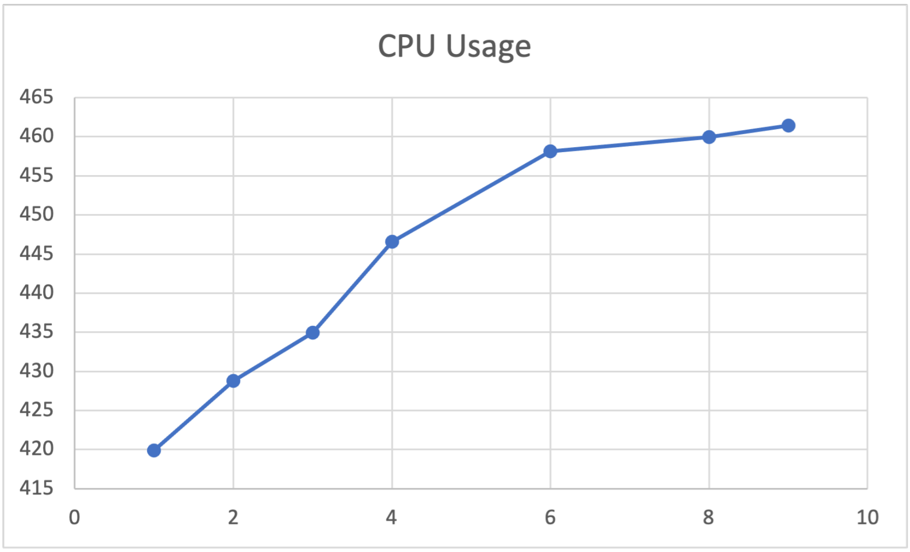


Figure 11: CPU vs Parameter

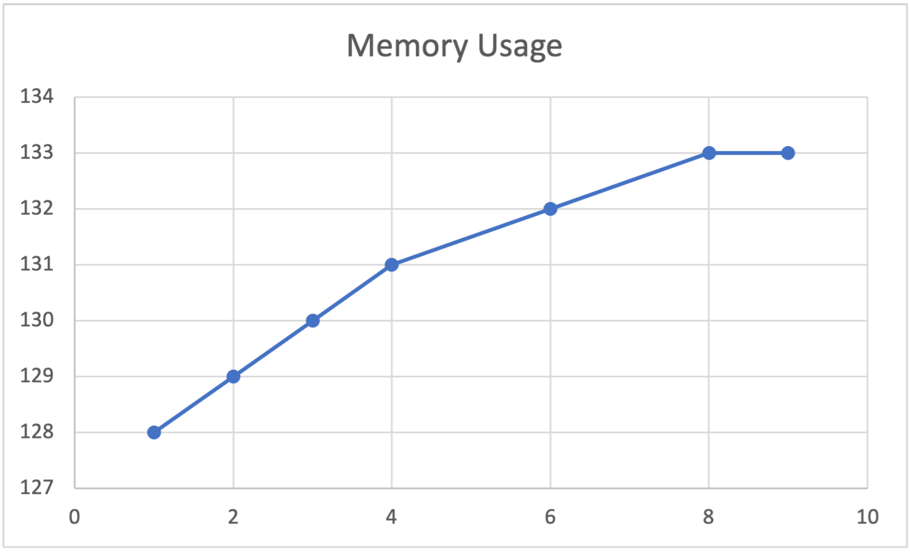


Figure 12: Memory vs Parameter

From the above table and graph, it is clear that as the number of threads are increased so does the memory and CPU Usage. Each thread forces some processing power from the CPU and memory. This shows us an increasing graph. However, in real life, hardware is limited. Hence, we see that the CPU usage gets saturated. However, the memory usage doesn't saturate. It rarely increases even. This is because at a time we are processing a single image, just that the image is broken into many pieces. There sum however is the same image.

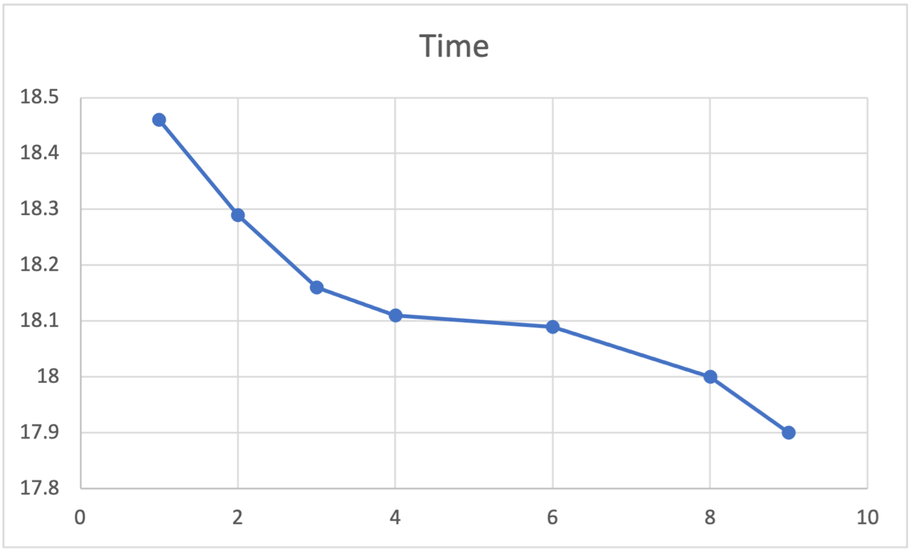
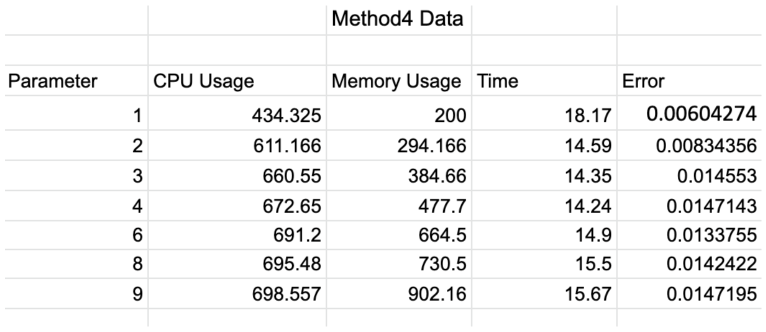


Figure 13: Time vs Parameter

There will be little to no Error in the result as we are processing all part of the frame. Just that we are processing it in parallel. Here is is evident from the plot that the run-time decreases as the number of parameters increase. This shows an inverse nature as expected. The run-time will increase with a large number of threads are there are limited core and memory and once overloaded, it becomes very slow, since each thread has it's own creation time, memory and power.

3.4 Method 4

Here the Work is spilt temporally. Frames are processed in parallel which makes each thread to process less frames and hence it makes then end result come faster.



		Method4 Data		
Parameter	CPU Usage	Memory Usage	Time	Error
1	434.325	200	18.17	0.00604274
2	611.166	294.166	14.59	0.00834356
3	660.55	384.66	14.35	0.014553
4	672.65	477.7	14.24	0.0147143
6	691.2	664.5	14.9	0.0133755
8	695.48	730.5	15.5	0.0142422
9	698.557	902.16	15.67	0.0147195

Figure 14: Table

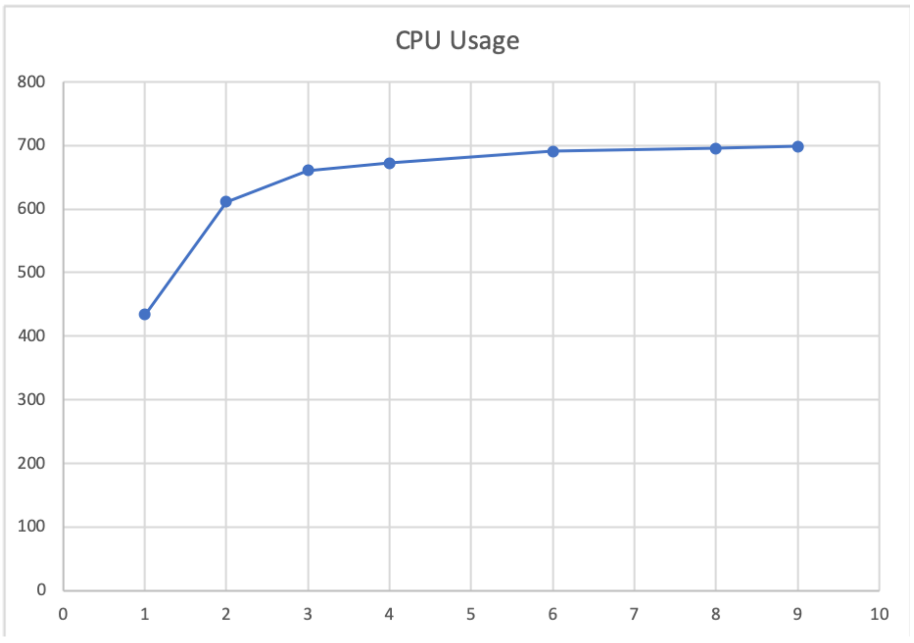


Figure 15: CPU vs Parameter

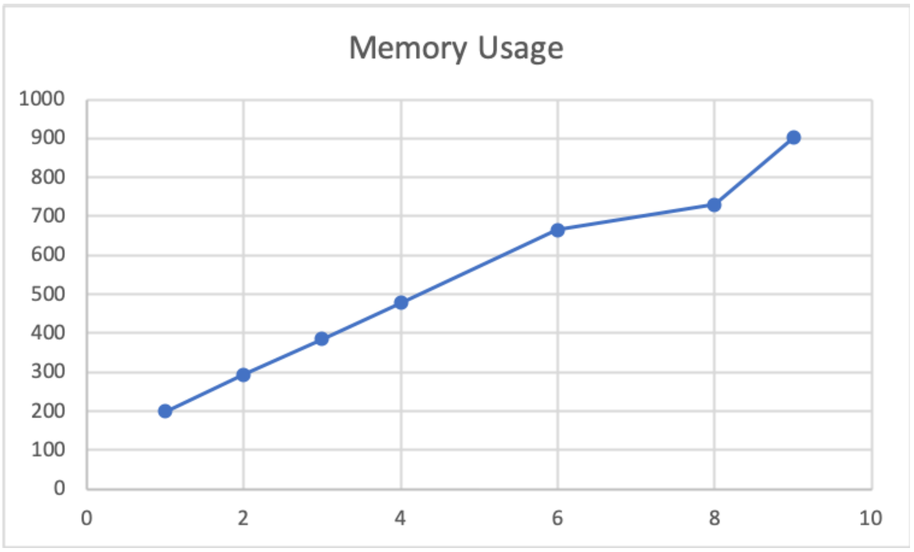


Figure 16: Memory vs Parameter

From the above table and graph, it is clear that as the number of threads are increased so does the memory and CPU Usage. Each thread forces some processing power from the CPU and memory. This shows us an increasing graph. However, in real life, hardware is limited. Hence, we see that the CPU usage gets saturated. However, the memory usage doesn't saturate, this is because we have much more memory compare to CPU power. To see memory saturation we would need to go to hundreds of threads.

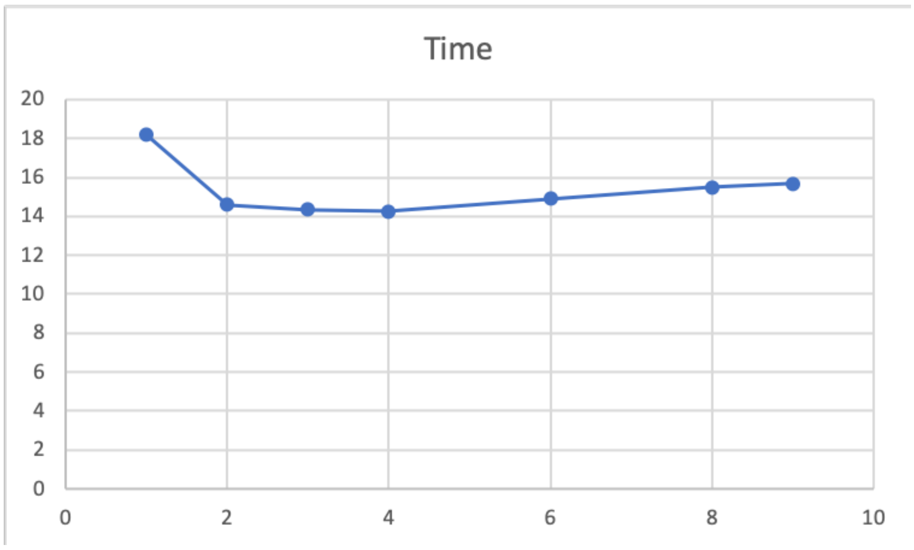


Figure 17: Time vs Parameter

There will be little to no Error in the result as we are processing the complete frame. Just that we are processing it in parallel. Here is is evident from the plot that the run-time decreases as the number of parameters increase. This shows an inverse nature as expected. The run-time will increase with a large number of threads are there are limited core and memory and once overloaded, it becomes very slow, since each thread has it's own creation time, memory and power.

4 Conclusion

We learnt about openCV functions and how to use them to carry out Computer Vision Tasks. We know how to process images and videos to extract meaning full information from them. We learnt Optical flow and background subtraction, and multi-threading in C++. Through the implementation of various methods across different parameters, we were able to perform a utility-runtime and runtime-parameter analysis. Finally, we conclude that each method has it owns advantages and disadvantages and according to various usages we could employ different methods.