COP290 Task 1

Rishi Shah
Harshil Vagadia

March 2021

1 Problem Statement

Analysing utility - runtime tradeoff on traffic density estimation task by implementing various optimisations

2 Design Choices

We have made the following design choices:

1. In queue and dynamic density calculation we have used absdiff method provided by the opencv. For queue density estimation, we take absdiff between current frame and the empty frame. The queue density is proportional to the difference. Similarly we take the difference between two consecutive frames for dynamic density estimation
2. We first convert each frame to grayscale so that we get a single pixel value at each location. We can directly calculate the sum of pixel values of the difference image to get a single number metric for queue and dynamic density.
3. The four corner of the roads (which were supposed to be taken as input from the user in part 1) are now hardcoded to maintain uniformity across all the experiments.
4. Our baseline which we submitted in part 2 takes only every fifth frame of the video to avoid noise (corresponding to x = 5 in method 1)
5. For plotting the graphs, the runtime is considered as the time required only to calculate the densities and not the preprocessing steps like storing the frames.
6. The whole process it automated. Our cpp code reads the parameters from JSON file, which gets modified by our script. Script is written in python. The graphs are made using pandas and plotly.
7. We are plotting *param* vs *error*, *param* vs *runtime* and *error* vs *runtime* graphs.
8. Our program allows multiple methods to be applied simultaneously as defined by the parameters. However, only one of the methods 3 and 4 can be used at a time.

3 Metrics

The error is calculated is the RMSE (Root Mean Square Error). The runtime is the time required to calculate the densities of all the processed frames of the video.

4 Methods

We have implemented all 4 methods in the problem statement.

1. In *method*1, we take x as the parameter. This means that only every xth frame will be processed.
2. In *method*2, we take *resolve* as the parameter. We decrease the resolution of the frame such that the final size of the frames will become $tot\_rows/resolve$ x $tot\_columns/resolve$.
3. In *method*3, we take *space_threads* as the parameter. We divided each row and column in *space_thread* parts. Each part is processed by a seperate thread and add the results of all the parts to get the final output. Here we have used *mutex* to lock and unlock a variable which was used by multiple threads.

4. In *method*4, we take *time_threads* as the parameter. We divide our video in *time_thread* parts and each thread works on a particular part. At the start of each section, we assume that the dynamic density is zero as we do not have a previous frame (This is slightly different then the method 4 decribed on course page. This is because we think our method will have a similar runtime, but a lower error and easier implementation).

5 Trade-Off Analysis

5.1 Method 1

1. Parameters : x = 5,6,7,8,9.
2. Graphs :

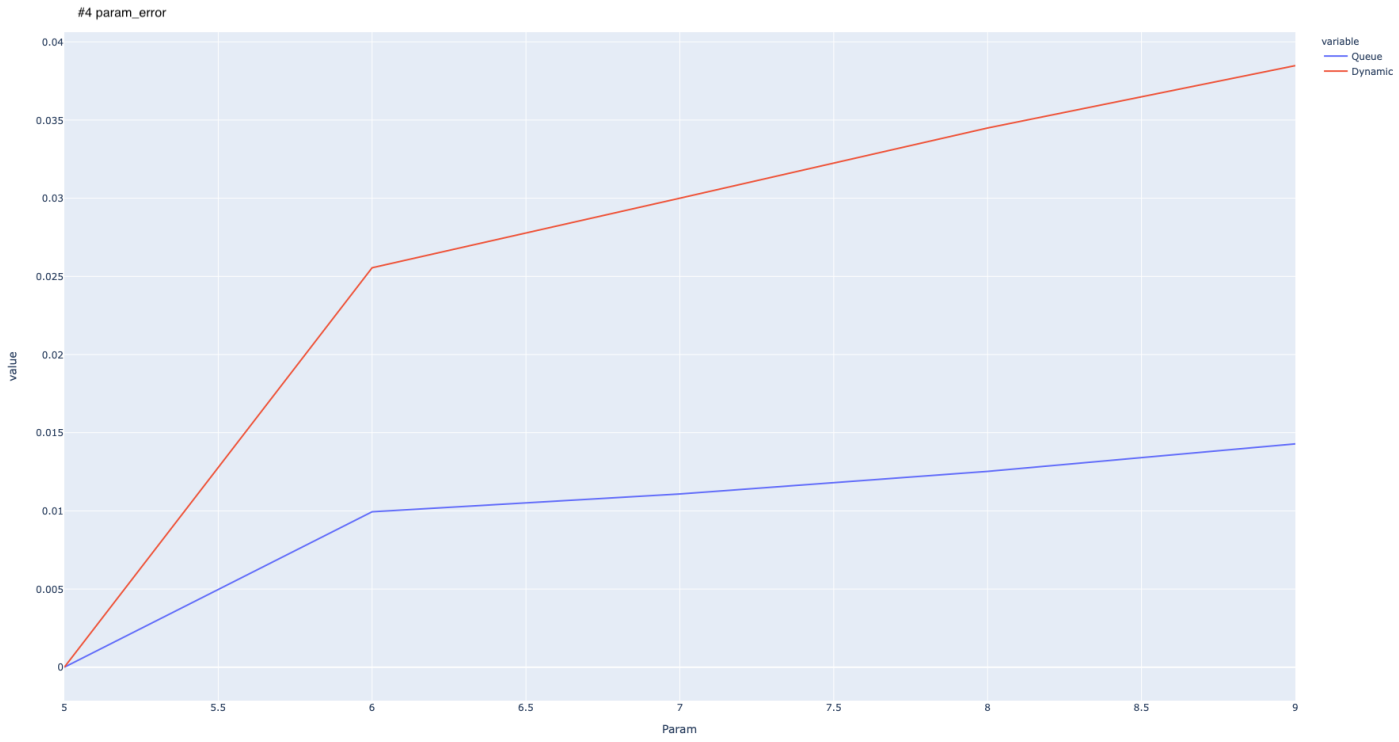

Figure 5.1.1: Parameter vs Error

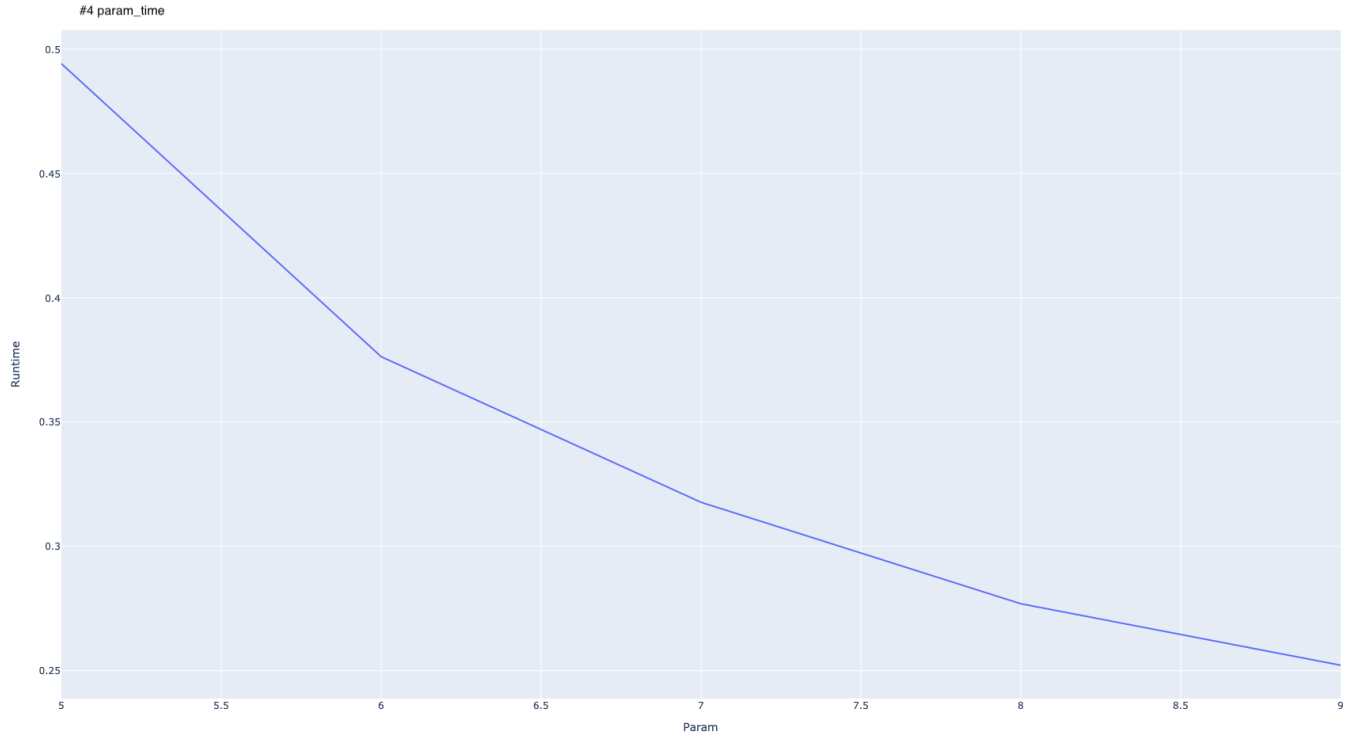

Figure 5.1.2: Parameter vs Runtime

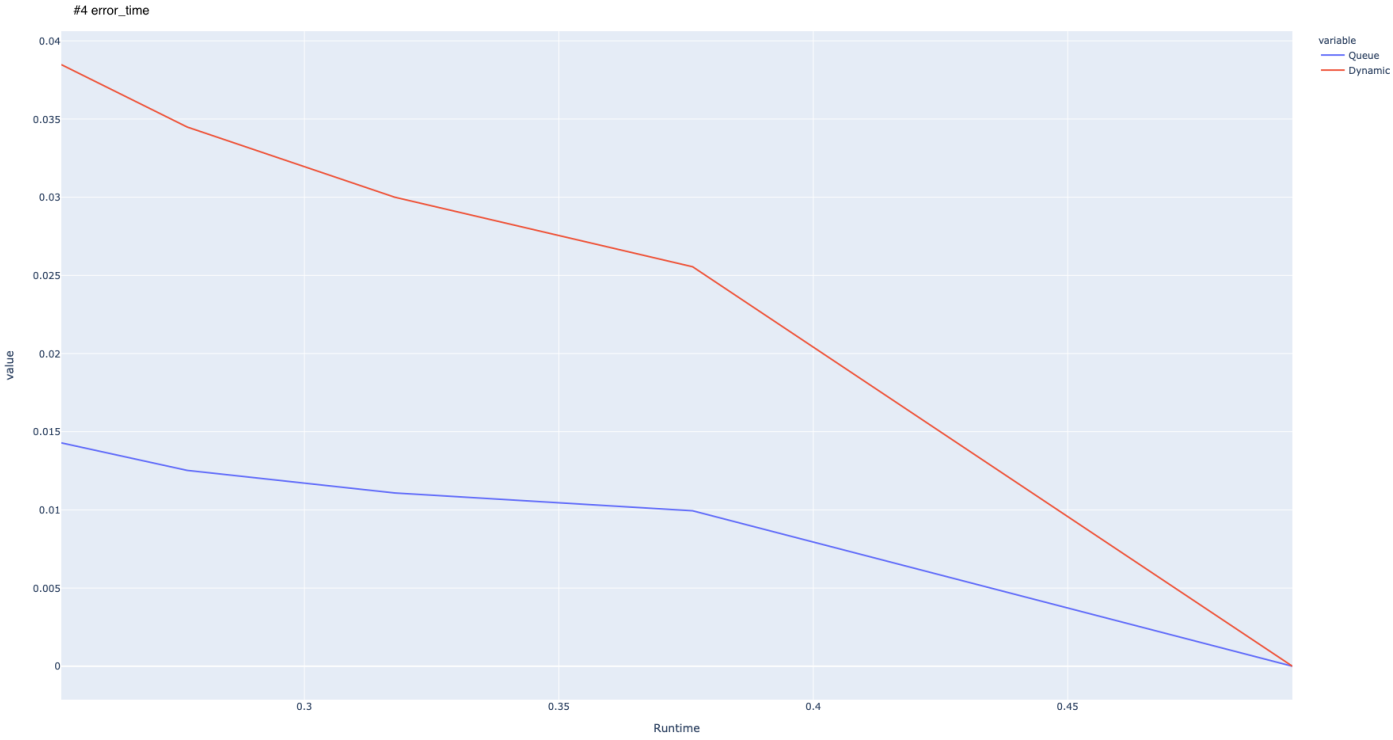

Figure 5.1.3: Error vs Runtime

3. Tables :

x	Error_Queue	Error_Dynamic	Runtime(sec)
5	0.000000	0.000000	0.49415
6	0.009942	0.02555	0.37629
7	0.011082	0.03000	0.31774
8	0.012527	0.03449	0.27694
9	0.014286	0.03848	0.25222

4. Explanation : As x increases, we skip more frames. Since less number of frames are processed, the runtime decreases. However we lose the information of the intermediate frames and hence the error increases.

5.2 Method 2

1. Parameters : $resolve$ = 1,2,3,4,5,6,7,8,9,10.
2. Graphs :

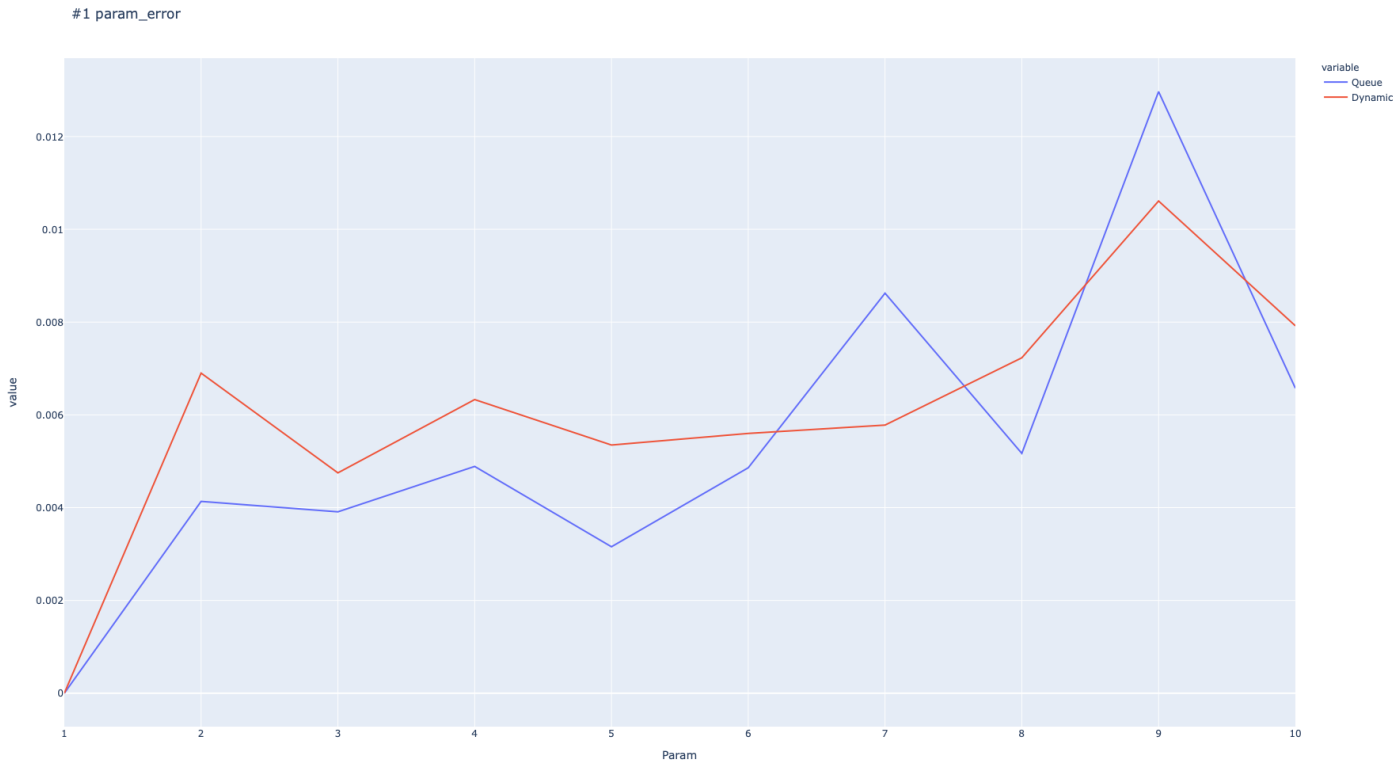

Figure 5.2.1: Parameter vs Error

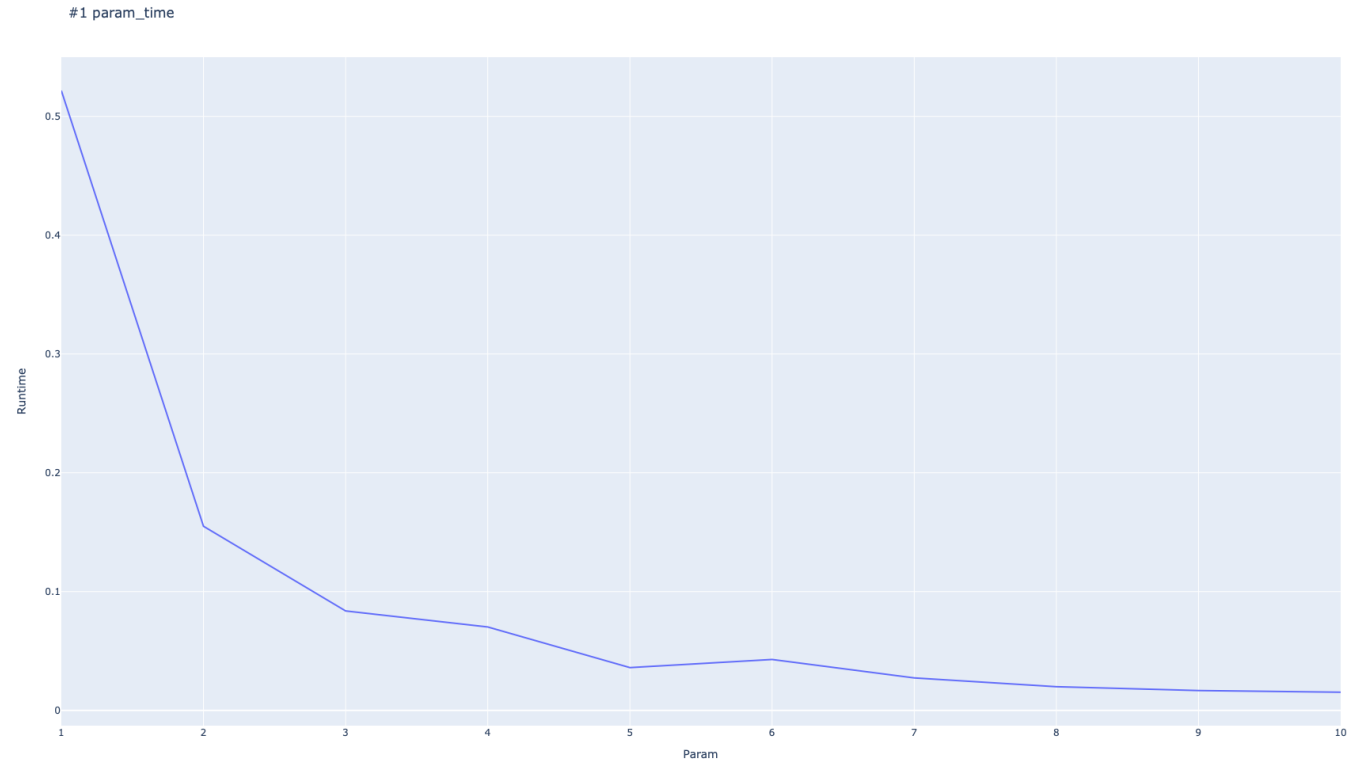

Figure 5.2.2: Parameter vs Runtime

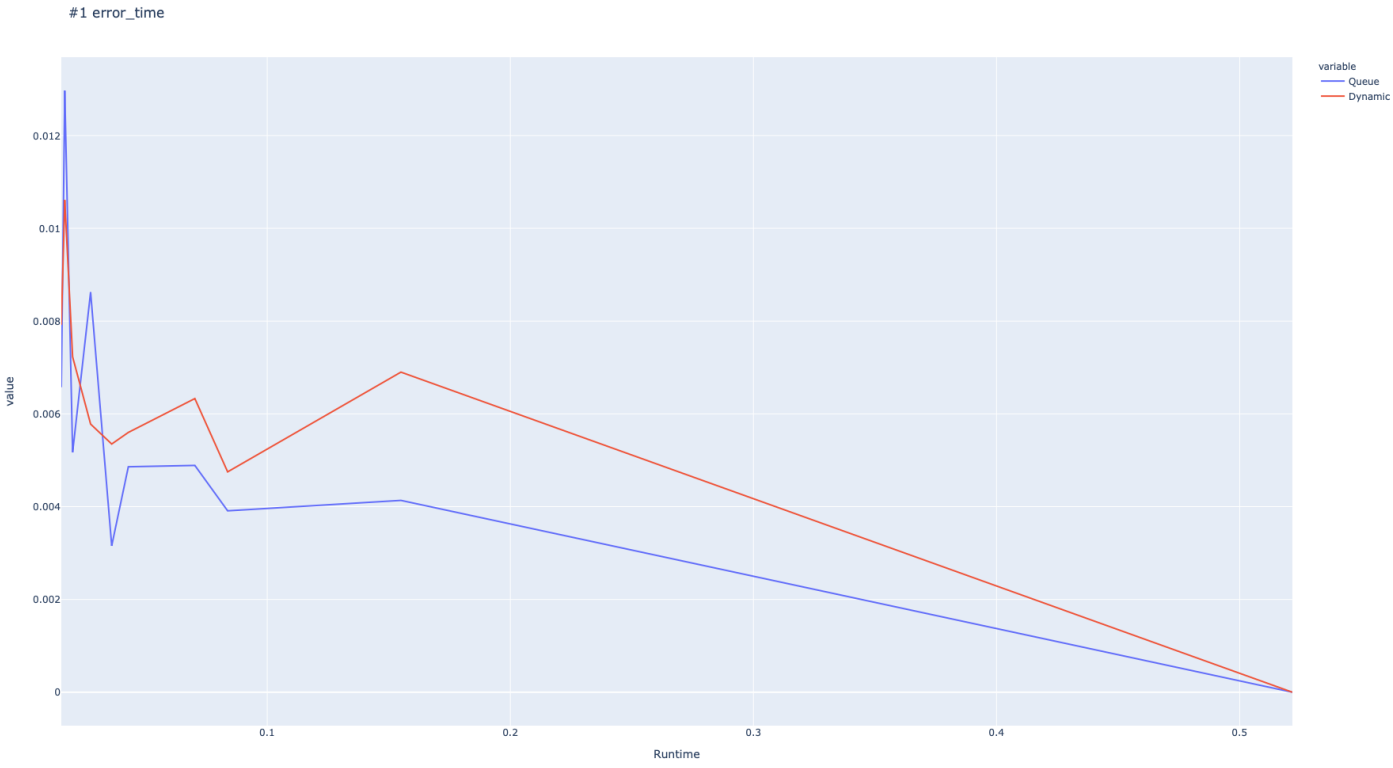

Figure 5.2.3: Error vs Runtime

3. Tables :

Resolve	Error_Queue	Error_Dynamic	Runtime(sec)
1	0.000000	0.000000	0.52168
2	0.004135	0.00690	0.15500
3	0.003910	0.00475	0.08372
4	0.004889	0.00633	0.07023
5	0.004860	0.00560	0.04287
6	0.003157	0.00535	0.03603
7	0.008623	0.00578	0.02735
8	0.005170	0.00723	0.01995
9	0.012968	0.01061	0.01672
10	0.006575	0.00792	0.01531

4. Explanation : As *resolve* increases, the size of every frame decreases. Since now we have to process a smaller frame, runtime decreases. The error increases due to the loss of pixels of each frame. However, the error is a bit noisy as the exact pixels lost depends on the value of *resolve*.

5.3 Method 3

1. Parameters : *space_threads* = 1,2,3,4,5.
2. Graphs :

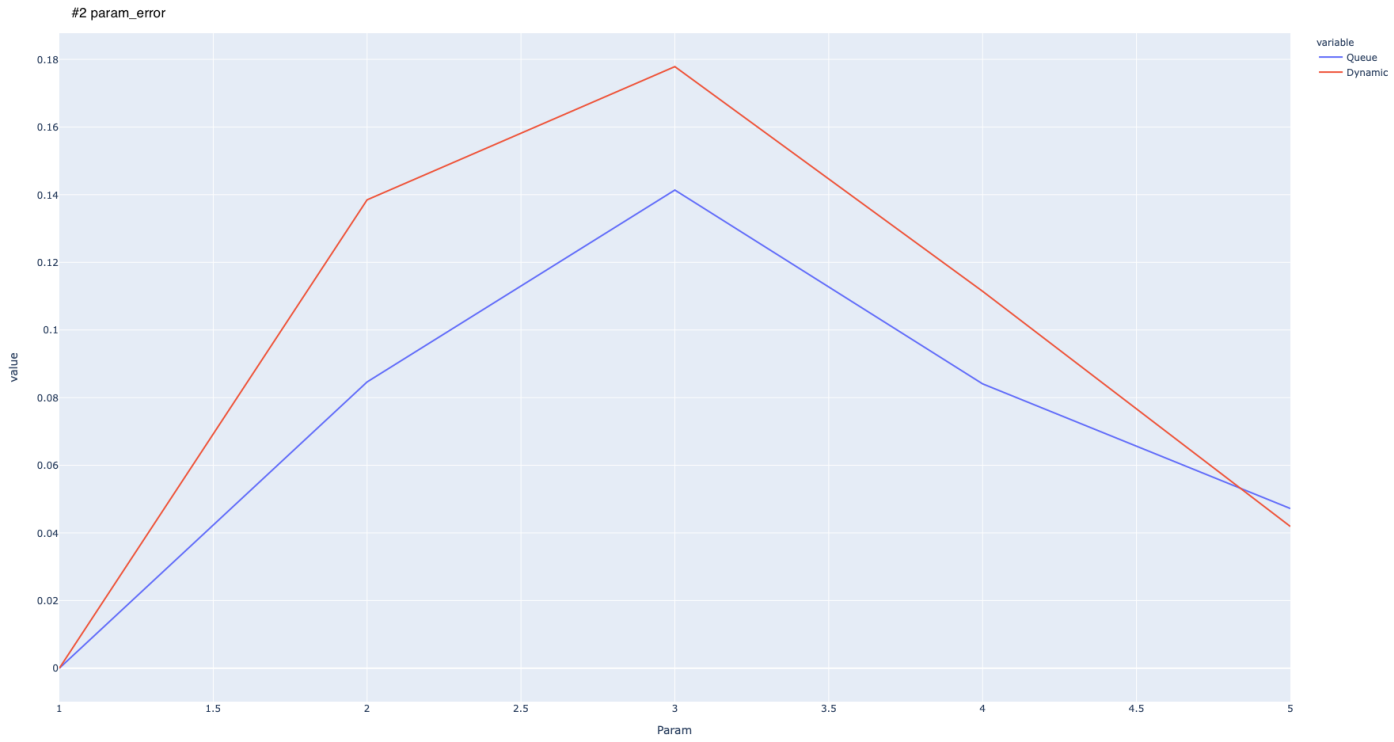

Figure 5.3.1: Parameter vs Error

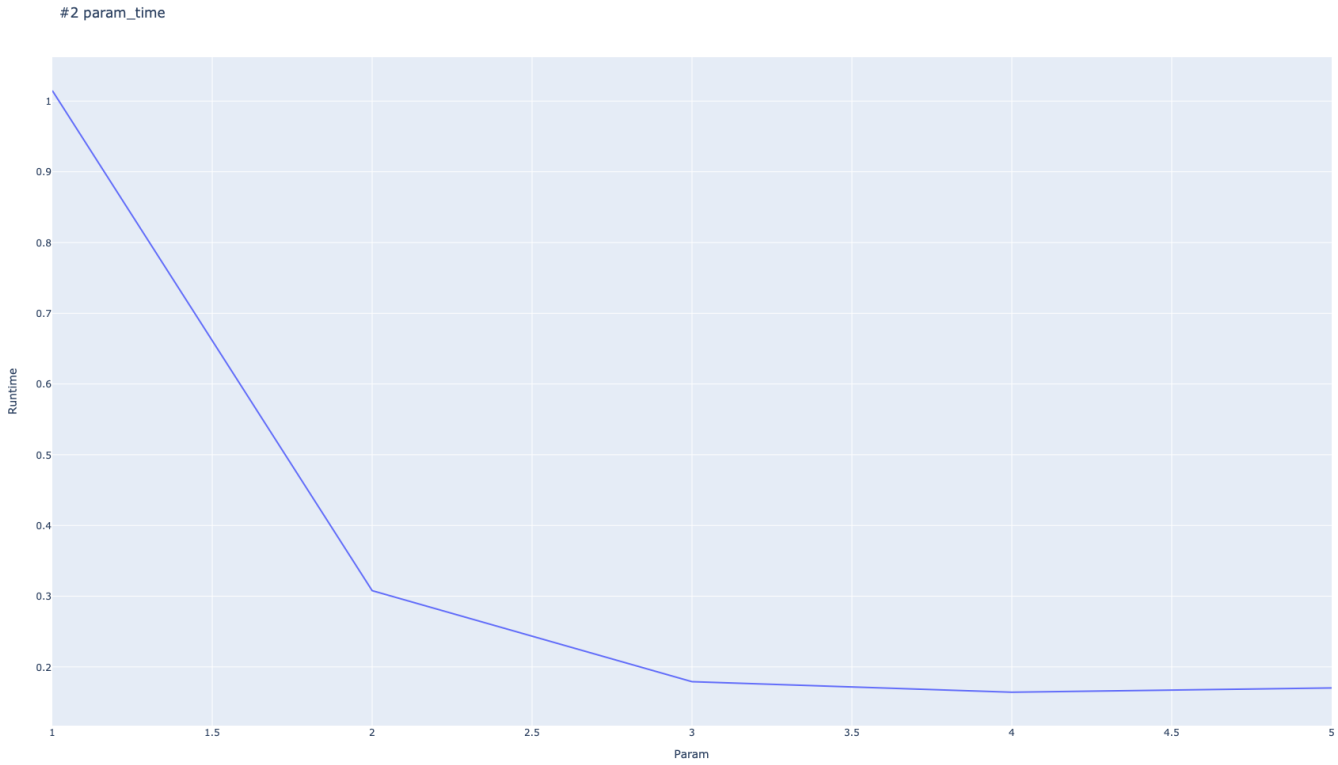

Figure 5.3.2: Parameter vs Runtime

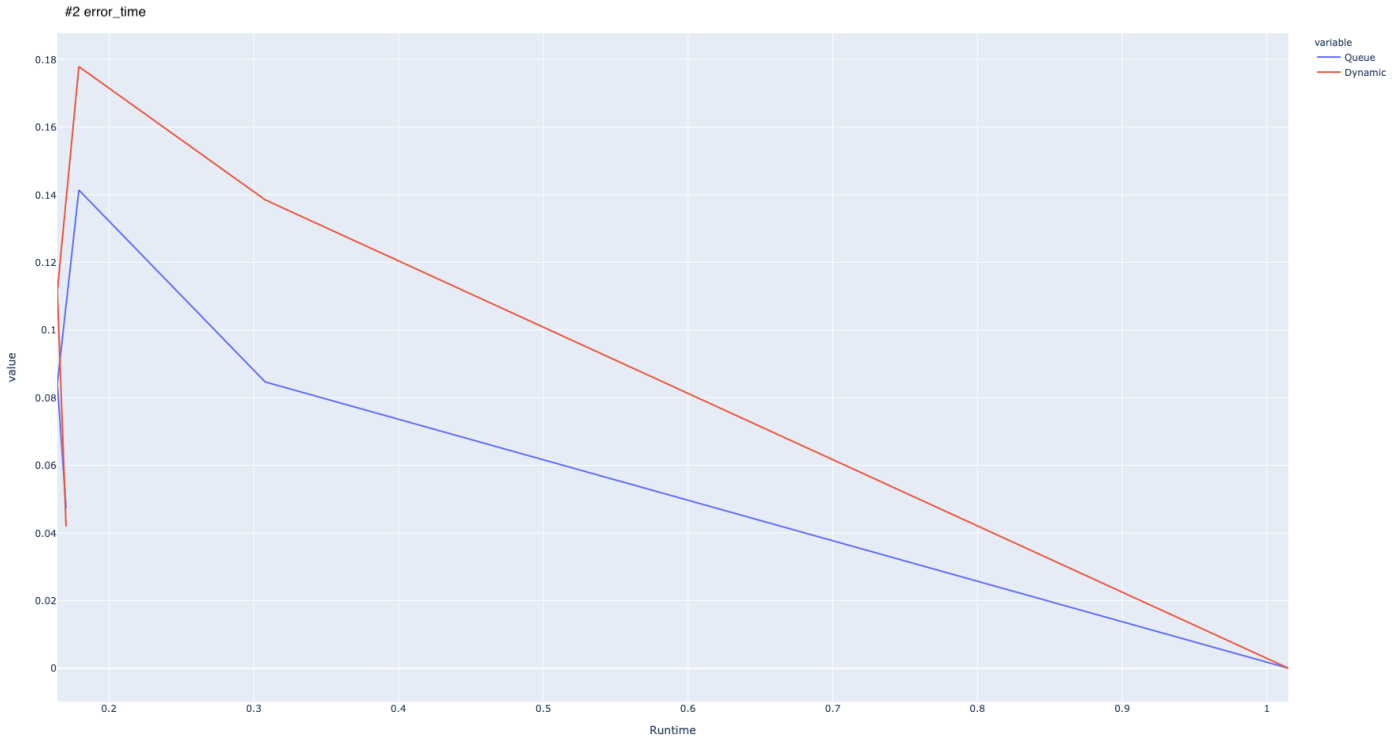

Figure 5.3.3: Error vs Runtime

3. Tables :

Space_Threads	Error_Queue	Error_Dynamic	Runtime(sec)
1	0.000000	0.00000	1.01477
2	0.084630	0.13847	0.30781
3	0.141363	0.17785	0.17911
4	0.084067	0.11143	0.16422
5	0.047211	0.04190	0.17027

4. Explanation : When we use space threads, each frame is being worked upon by more threads. Since each thread processes a smaller portion of a frame, the runtime decreases. It is difficult to predict the error as it depends on the relative signs of the difference image portion.

5.4 Method 4

1. Parameters : $time\_threads$ = 1,2,3,4,5,6,7,8,9,10.
2. Graphs :

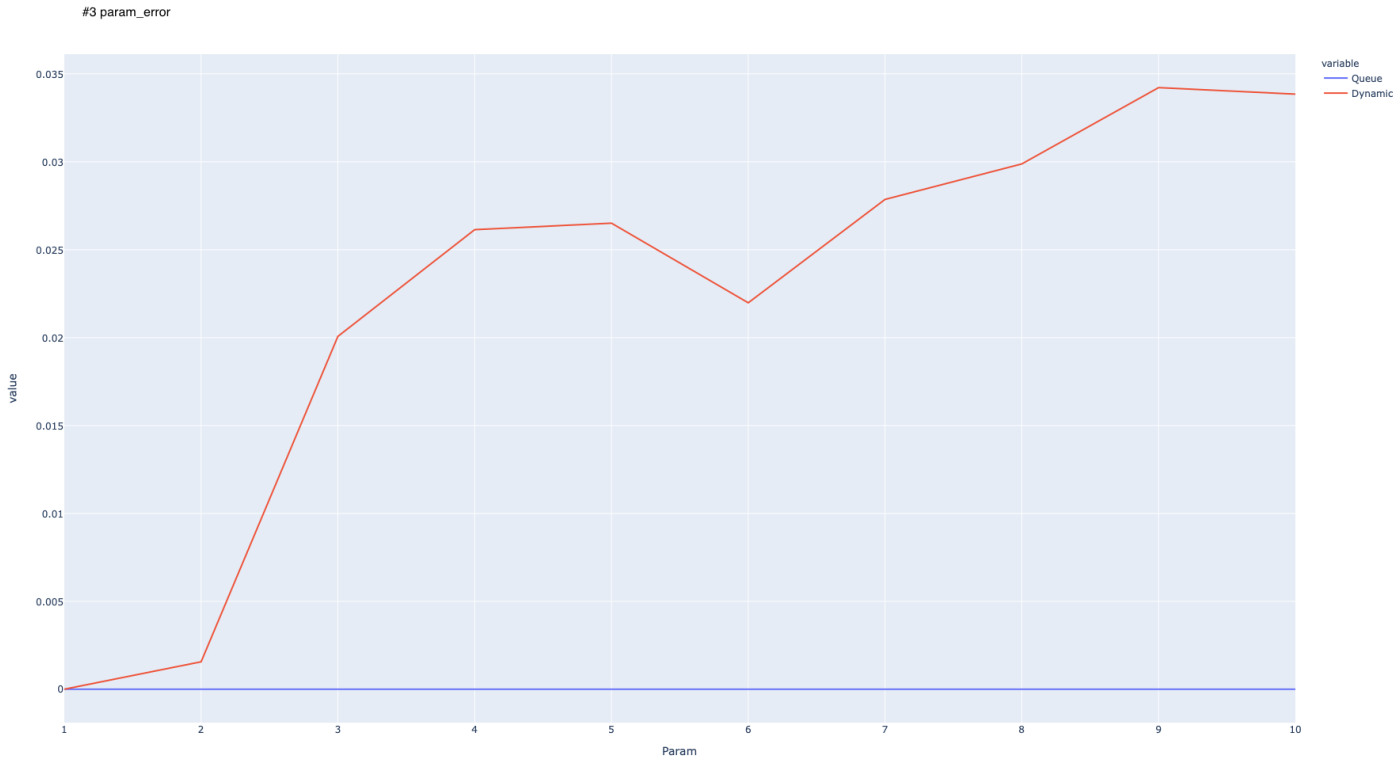

Figure 5.4.1: Parameter vs Error

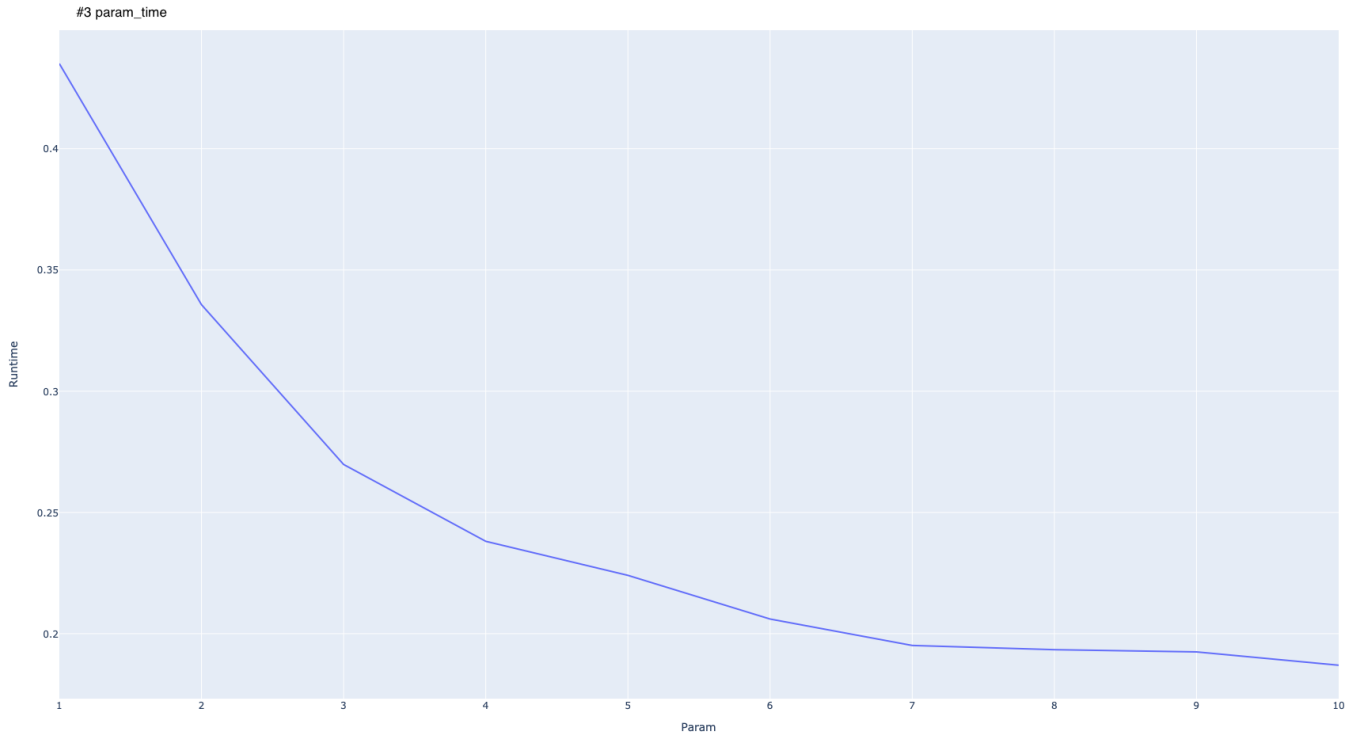

Figure 5.4.2: Parameter vs Runtime

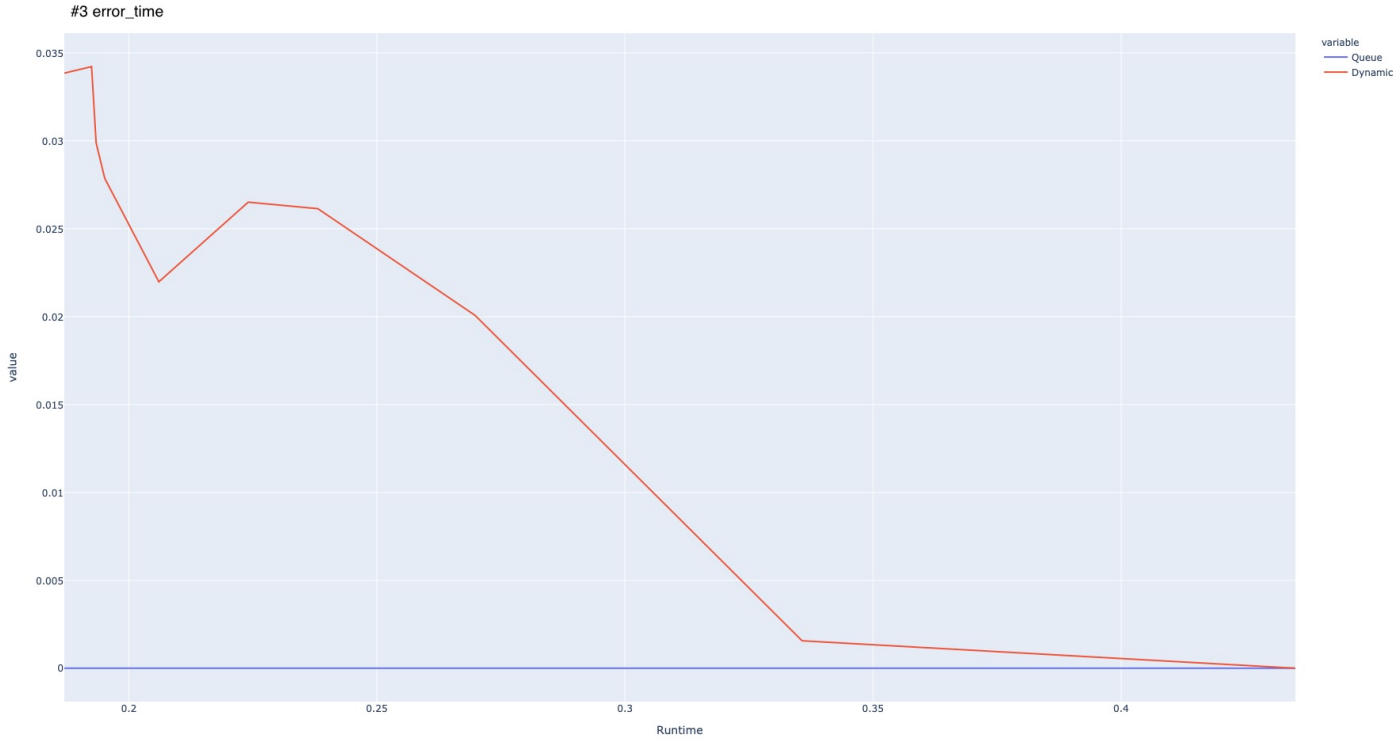

Figure 5.4.3: Error vs Runtime

3. Tables :

Time_Threads	Error_Queue	Error_Dynamic	Runtime(sec)
1	0.0	0.00000	0.43511
2	0.0	0.00156	0.33569
3	0.0	0.02007	0.26978
4	0.0	0.02614	0.23809
5	0.0	0.02651	0.22405
6	0.0	0.02198	0.20604
7	0.0	0.02786	0.19514
8	0.0	0.02988	0.19340
9	0.0	0.03422	0.19248
10	0.0	0.03385	0.18699

4. Explanation : As *time_threads* increases, the video gets broken in more parts. Since each thread processes a smaller portion of the video, the runtime decreases. However error increases in dynamic queue because of corner case adjustments.

6 Results

Near to expected graphs and results were obtained. Queue and dynamic densities are properly estimated.