COP290 Assignment 1

Navneet Jindal, Anirudh Gupta

March 2021

1 Introduction

This document contains a report on the analysis for different methods implemented to find traffic queue density and traffic dynamic density. The analysis is made from calculating utility i.e. average absolute difference of outputs of different methods from the baseline. The baseline is taken as the code of output found in subtask2 which finds both queue and dynamic density. The code written in method1 and method2 is compared from this file. There is code written separately for only queue density. For this case the baseline code is written in a file sub2queue.cpp which considers only queue density. The code written in method1-static and method2-static is compared with this as baseline. Metrics used in below analysis are utility and execution time.

1. Utility : The avg absolute error of traffic density found in comparison to baseline.

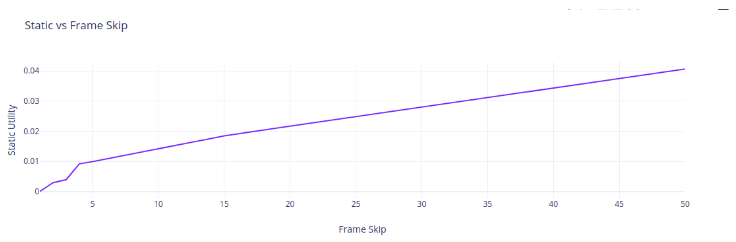
2. Execution time : Time taken to find the traffic density. It starts after choosing the source points.

2 Method1

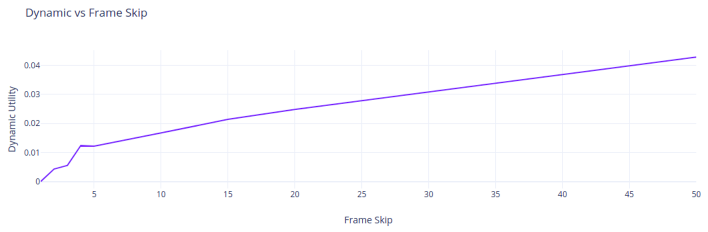
Here we skip certain number of frames to find the traffic density. In the skipped frames we assume traffic density to be the last calculated one. In calculating the dynamic density, we consider the just previous frames even though they were skipped but we do the homography of only needed frames.

In this analysis we have build three graphs (x - the number of frames we will skip + 1)-

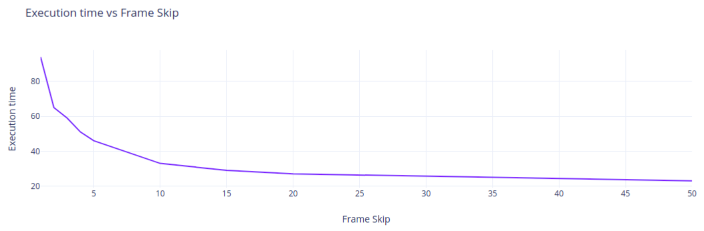
1. Static Utility vs x



2. Dynamic Utility vs x



3. Execution time vs x



Here we notice that as we increase the value of x the execution time decreases (because we have now processed lesser number of frames) while reaching to an asymptote. This happens because even if we skip a lot of frames in calculating static and queue density, it will still read every frame at least once whose time dominates for large values of x.

Utility increases with increase in x as we are now not calculating dynamic and queue density for skipped frames and taking the last calculated value as its own. For the frames for which we have calculated the density gives the same result as the baseline code for both static and dynamic density as we have considered the 'just' previous frames in finding dynamic density even though we skipped calculating density for those frames. Also note that we did homography of only those frames which were needed in calculating dynamic density.

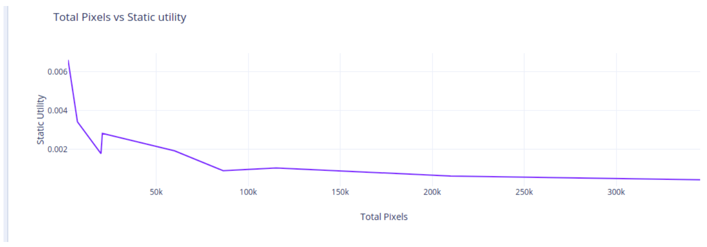
Another thing to note is that the utility increases linearly with x. This is because the number of frames with error increases linearly with x (each frame with same error on an average).

3 Method2

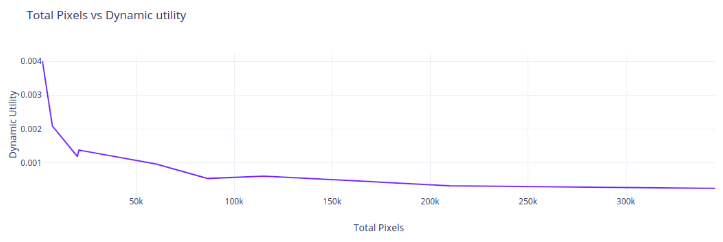
In this method we decrease the resolution of the cropped image which we process to find the traffic density using background subtraction. Here we have compare the results with total pixels (x*y) present in the final resolved image.

In this analysis we have build three graphs ((x,y) - resolution of image)-

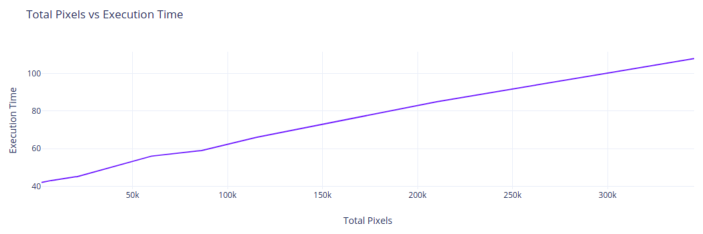
1. Static Utility vs x*y



2. Dynamic Utility vs x*y



3. Execution time vs x*y



Here we notice that as we decrease the value of x*y the execution time decreases. The execution time decreases basically due to decreasing order of the nested for loop in background subtraction. Also it decreases linearly proving that the order of the algorithm depends linearly on the total number of pixels. Also we can see that the graph's y-intercept is non-zero which basically gives the time taken to read the whole video.

Here utility shows an interesting point. Utility is greater than zero because when we resize our image in less resolution we lose a lot of information. In this case utility can both increase or decrease. As we decrease x*y by large margins utility will start to increase as we our then losing a lot of information in subsequent tests. But if we consider any two resolutions which are near to each other than it doesn't necessarily mean that one with more pixels will have lesser utility. This is because both of them lose information but in what way it affects the result depends upon the image itself (as we can see a slight increase in utility in between).

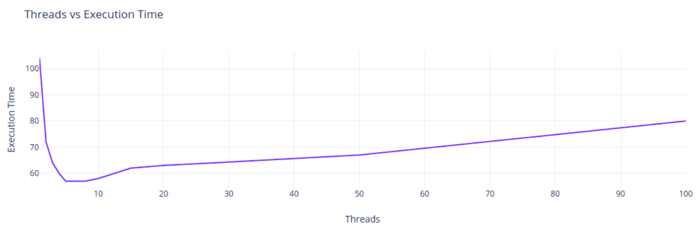
There are various methods to reduce the resolution of image. Experimentally we found that INTER-NEAREST method of opencv gives the least utility, so we have used the same.

4 Method3

Here we try to implement multi-threading to process each frame faster in comparison to before. In background subtraction we iterate through two nested for loops. By using multi-threads we try to run it faster by running different segments of for loop in different threads.

In this analysis we have build one graph (x - the number of threads used)-

1. Execution time vs x



Here we notice that as we increase the value of x the execution time decreases while reaching to a minimum before increasing again. This happens because of limits of the hardware system we have. Even with no hardware limitations we won't be able to reduce the execution time below a certain threshold. This is because reading every frame is still done by the main thread whose time dominates over the time taken by large numbers of threads.

Here utility turns out to be zero for both static and dynamic cases as we have not done any approximation in calculating the density. We have just run different segments of the nested for loops for background subtraction parallel using threads, thus decreasing the execution time.

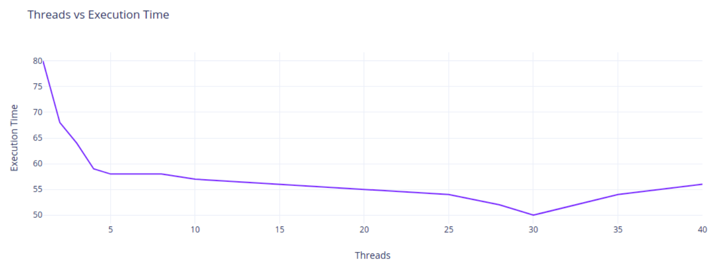
As we increase the number of threads by large values we see an increase in execution time. This is because of hardware limitations. Although we have increased the number of threads the system still has the same number of cores trying to run more threads and the main thread has to create more threads increasing the execution time.

5 Method4

Here we use multiple threads to run many frames in parallel to each other. We give 16 consecutive frames to one thread to process.

In this analysis we have build one graph (x - the number of threads used)-

1. Execution time vs x



Here we notice that as we increase the value of x the execution time decreases while reaching to a minimum before increasing again. This happens because of limits of the hardware system we have.

Here utility turns out to be zero for both static and dynamic cases as we have not done any approximation in calculating the density. We have given set of consecutive frames to different threads to process for background subtraction which happens parallelly, thus decreasing the execution time.

As we increase the number of threads by large values we see an increase in execution time. This is because of hardware limitations. Although we have increased the number of threads the system still has the same number of cores trying to run more threads and the main thread has to create more threads increasing the execution time.

Some peculiar observation :

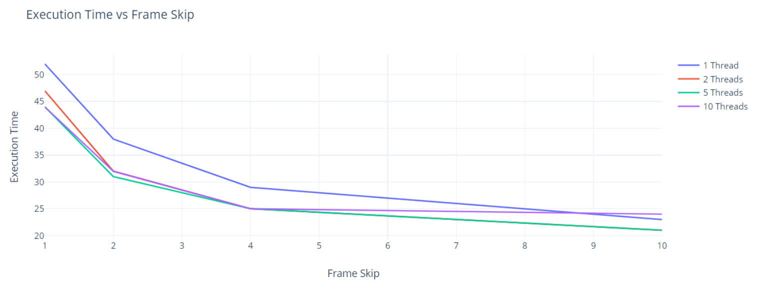
First we implemented this method by giving consecutive frames to different threads. In this we observed the time reduction was not much. Even the time taken by only one thread was approximately the same as time taken by 4 threads. This was because when we gave consecutive frames to different threads, a thread operates on frames which are not consecutive. This increases time taken for one thread to access the frames because they are not in system cache. In contrast to this we now gave 16 consecutive frames to each thread. Here it takes less time for the thread to access the same number of frames as they are now present in system cache memory.

6 Method1-static

Combining method1 and method3.

In this analysis we have build one graph (x - the number of frames we will skip + 1) for different number of threads.-

1. Execution time vs x



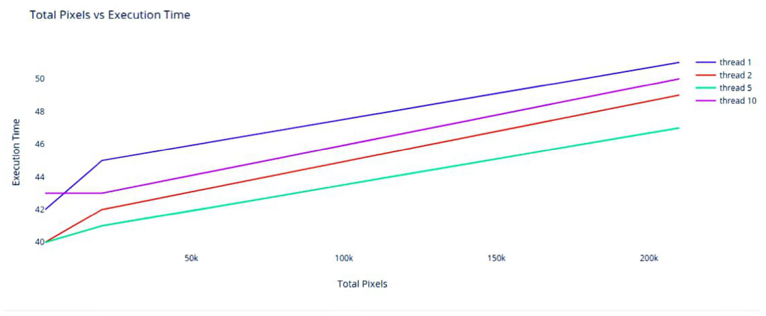
Here we study the effect of method1 and method3 simultaneously. The results of this part is similar to method1 and method3. Here the execution time decreases with more frame skip, also same happens with more threads until a specific number of threads before increasing again. We can also compare the effect of threads and frame skip. Here we observe that frame skip is more powerful than working with threads (but error is also greater in the same). As for utility, it yields the same results as in method1.

7 Method2-static

Combining method2 and method3.

In this analysis we have build three graphs ((x,y) - resolution of image) for different number of threads-

1. Execution time vs x*y



Here we study the effect of method2 and method3 simultaneously. The results of this part is similar to method2 and method3. Here the execution time decreases with lesser number of pixels, also same happens with more threads until a specific number of threads before increasing again. We can also compare the effect of threads and total pixels. Here we observe that both have nearly the same effect (where method2 gives error and method3 has hardware limitations). As for utility, it yields the same results as in method2.

8 Sparse vs Dense optical flow

Dense optical flow or Farneback's method to detect motion considers two consecutive frames of a video and detect every motion which takes place in going between those two frames. Sparse optical flow is a lot different. It first finds good features worth tracking (corner points of a vehicle) and then checks their movement in two consecutive frames. Here we can see that it detects motion for only some specific points giving it higher speed of execution but less accuracy.Thus a scaling factor is needed (here we took it to be 50 which was found experimentally) to normalize the traffic density to undertake the width of a vehicle. The execution time for dense optical flow comes out to be 393 sec and while the same for sparse optical flow is 72 sec. But it also increased the error to 0.2.