Runtime Tradeoff in Traffic Monitoring Embedded Systems

Harsh Agrawal
2^{nd} Year UG Student
Dept. of Computer Science and Engineering
Indian Institute of Technology, Delhi
New Delhi, India.
cs1190431@iitd.ac.in

***Abstract*—In this report, we have analyzed the effect of various computational parameters, and their effect on accuracy on the computer vision based traffic monitoring systems.**

I. Introduction

Recent Developement of Smart Cities, has resulted in increase in systems capable of monitoring road traffic for both environmental analysis, and better traffic management. This requires solutions such as embedded systems, doing live inference from camera feed of CCTV cameras, deployed across many highways, expressways and road junctions in city. However these embedded systems, are limited on computational capacity, and may not have a large source of energy(mostly relying on solar energy). Also based on the time of day and day of the year, the varying temperatures, add a restriction on how much computational power they can spend without getting overheated. In this report we investigate the various computational parameters, that can be tweaked, such that the computational load on embedded systems decreases, while maintaining the performance as much as possible. Since we do not have Ground Truth Data, we create a baseline model, which practically takes the largest time, but is assumed to be most accurate. All other models are evaluated against this baseline model. We use Mean Squared Error to analyze the tradeoff between time taken and utility of model. The following sections are divided as follows: In the next section we will discuss the basic algorithm of model developed by us to calculate traffic density. This is followed by a section on our approach to study the effect of runtime and utility. Further in Section 4, we develop a framework, that can assist these systems to take dynamic decisions based on ambient conditions. In Section 5 we discuss the overall results, after which we conclude in Section 6.

II. Algorithm

We run our algorithm on a real footage from CCTV camera installed at Lajpatnagar junction in New Delhi. [2] The camera has a wide range and is also doesn't gives a birds eye view. To solve this, we initally crop each of the frame and correct its perspective so as to get a birds eye view. For perspective transform, we use OpenCV's 'findHomography' and 'warpPerspective' algorithm. [4] The region to be cropped for a junction is fixed (given the orientation of camera) and is taken from the user or is hardcoded. Then, the background from this image is removed using MOG2 Algorithm. The Algorithm is either trained on empty photo of road or using first few frames of the input video. The image is then processed through a series of dilations and erosions, after which we perform binary thresholding. This creates a map of all the vehicles, and using this we calculate the **queue traffic density**. In order to calculate the dynamic density, we deploy two models, one uses Gunner Farneback's algorithm for solving optical flow equation, while other is Lucas-Kanade's optical flow algorithm. [1] [3] While the former one is slower than the latter, it gives better results. These algorithms can be switched depeding on the computational power available.

III. Metrics

To define our utility we calculate the Mean Squared Error from the baseline set of parameters. This error is average over the time of video. To define the runtime, we evaluate the time taken by given video(Size: 382s). To ensure results are not varied due to other tasks on computer, program runs are made on Linux Environment without explicitly running any additional software. For each set of parameters, 3 runs are performed, and their average is taken to further ensure that results are not pertubed by external factors.

IV. Methods

To study the effect of runtime and utility we consider five optimizations:

- Frame Sub-Sampling
- Image Resolution Reduction
- Multi Threading (Spatial)
- Multi Threading (Temporal)
- Using Sparse Optical Flow

In this section effect on utility for each of these is studied independently. The metrics are varied and corresponding utility is observed and plotted. Analysis of each of these method is included in the following subsections itself.

A. Frame Sub-Sampling

In this Method, we only take every xth frame of the video for processing. For the skipped frames, we interpolate the

density values from previous frames. We observe that as we increase the the number of frames skipped, the time decreases, however the error continously increases [See Fig.]. However we also note that change in runtime is not much after skip frames is made equal to four, since loading frames from video starts having larger relative effect. Thus we can say for given model, setting skip_frames $\bar{6}$, is best when running on limited performance, as it is able to improve the runtime by a factor of $\approxeq 2.5$.

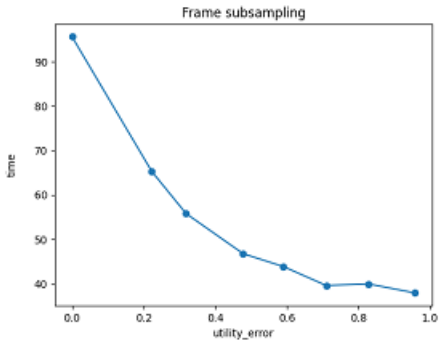

Fig. 1. We observe no suitable advantage on skipping more than 6 frames is observed.

B. Image Resolution Reduction

In this Method, we reduce the resolution of each frame by a factor. The conversion is done in realtime, and all the processing takes place on this downscaled image. Unlike frame subsampling, we observe a linearly decreasing graph. However the difference in error in both of these graphs is significant, with frame subsampling outperforming the resolution method.

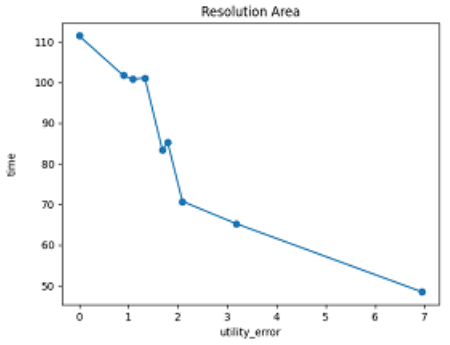

Fig. 2. An almost linear increase in utility error with decrease runtime is observed

C. Multi-Threading

Initially our code was single threaded executing statements sequentially, but there are lot of independent processes that can be executed in parallel to improve the execution time. We introduced multi threading in our code to improve the performance by utilising all cores of a multi-core processor. We have implemented a single producer, multiconsumer threading model using `semaphores`. A single producer reads frame from the input video file and places the frame into a shared buffer. In a multi-threaded environment several consumer threads are spawned which wait for the producer and start processing the frame. `semaphores` are used for thread synchronisation and to prevent race conditions among threads for shared memory.

1) Spatial: We split a frame into segments and assigned each segment to an independent thread. Splitting a frame reduces the matrix size required for each computation and reduces the average processing time of a thread. However this method also introduces considerable amount of errors since there is no thread that operates on the borders. Also we observe that increasing threads upto 4 has some advantage, however after that the runtime also starts growing. One of the reasons is that maintaining large number of threads, has its own overhead. Moreover, since the sizes of images become small, the runtime advantage of vectorised algorithms is lost.

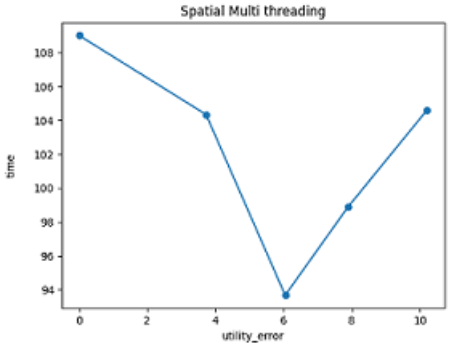

Fig. 3. We observe to get the least time, 4 threads are needed.

2) Temporal: We split the work temporally by giving consecutive frames to different threads for processing. The analysis was parametrized by the number of threads used for processing. This method proved quite effective and practically feasible. As the number of threads were increased the processing time decreased without any significant loss in utility. However there was no runtime improvements after more than 4 parallel threads were used. The reason for this is that processors have limit on number of threads they can use in parallel, and OpenCV algorithms being already heavily vectorised, the saturation is reached early.

D. Sparse Optical Flow

In this method we implement the sparse optical flow method for dynamic density calculation instead of the dense optical flow method. We observe a significant reduction in time taken(Roughly 1.8 times). However the error observed is higher than the other methods. The reason for this behaviour is partly due to the ability of sparse method to give zero output when there is actually no movement in road. This is in contrast to dense optical flow method, where there is always some noise. Thus it is fair to say the error from GroundTruth is sufficiently less while using sparse optical flow, and the

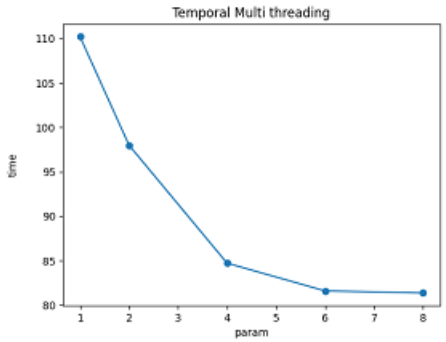

Fig. 4. No sufficient advantage is seen after increasing parallel threads to more than four

time gain is also good. Thus it is a good method to keep computational load in check.

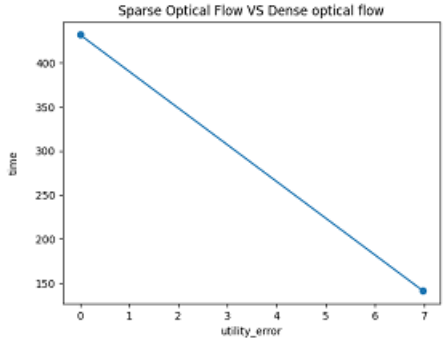

Fig. 5. While decrease in runtime is substantial, we observe error is substantial.

E. Dynamic Tuning

For embedded systems deployed in production it is necessary that they adapt to the enviornmental conditions. Also traffic monitoring systems should adjust to the traffic conditions, i.e when traffic is low, these embedded systems may move to low power mode. However, it should be noted that facilitating the same is a non-trivial problem. The reason is that different parameters do not correlate with each other in terms of error. Also no mathematical function exists to correctly map given error to runtime or vice-versa. To make this dynamic tuning possible, we run the program on wide combinations of tunable parameters and create a databse from the same. Now using a simple c++ function call, based on the error/time constraint, the best metric satisfying the given constraints with the best performance/least error is returned. This best metric is searched from the databse we have created. For example in a real system, a temperature sensor can send information to tune down performance by x% and the c++ function, will return the parameters to be used for least error.

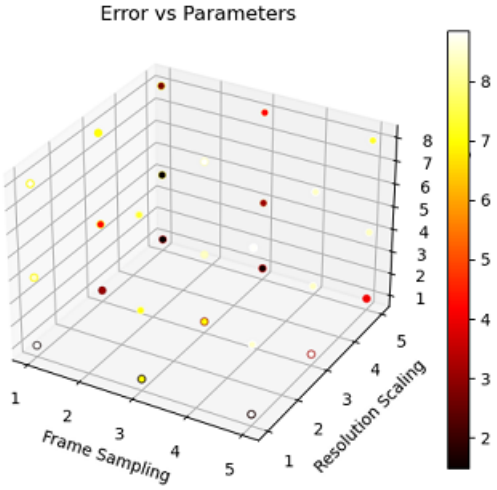

Fig. 6. Here z-axis denotes the number of temporal threads used.

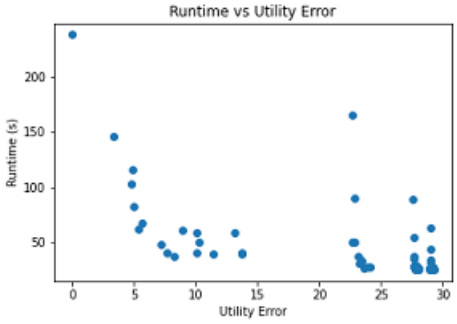

Fig. 7. The parameters corresponding to bottom left portion of graph are best to use when on a low computational budget

V. Trade-Off Analysis

For the various methods, we observe that all have their own advantages. However we note that Frame subsampling is able to reduce the computational load significantly, while seeing the least decrease in error. Also while the temporal multithreading is able to reduce runtime with almost no error, it is still computationally expensive, because of use of multithreading. At the same time multithreading's full advantage is not observed as OpenCV algorithms are already vectorised and optimised, leaving little scope for improvements in user created threads. In this report, we also studied the net effect of adjusting various parameters. We observe that parameters are not additive. For example effect of sparse optical flow on lower resolution in terms of time is negligible, however it induces much more absolute error than when on high-res images. Infact, the best compute friendly parameters are without sparse optical flow [See Fig. 7]. Overall we observe choosing the right set of parameters, can decrease computational time significantly (as high as 8-9 times) with minimal error.

VI. Conclusion

In this report, we have studied the effect of tuning parameters on runtime parameters for real-time computer vision based traffic monitoring system. We see most methods are able to reduce the runtime significantly, with a small penalty in accuracy. We observe that while some methods(frame subsampling), are able to outperform other methods in almost all cases, to improve the overall runtime, right combination of multiple methods is needed. However, if only one method is to be used, adjustign frame subsampling to improve runtime is the best method. One reason for this is that traffic densities change in scale of few seconds and thus even decreasing video to sat 2 FPS, cannot hurt the calculations.

References

[1] Gunnar Farnebäck. "Two-frame motion estimation based on polynomial expansion". In: *Scandinavian conference on Image analysis*. Springer. 2003, pp. 363–370.

[2] *LajpatNagarJunction Traffic Video*. 2021. URL: https://www.cse.iitd.ac.in/~rijurekha/cop290_2021.html.

[3] Bruce D Lucas, Takeo Kanade, et al. "An iterative image registration technique with an application to stereo vision". In: Vancouver, British Columbia. 1981.

[4] OpenCV. *OpenCV Warp Perspective*. URL: https://docs.opencv.org/master/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87.