Design Description

Navneet Jindal, Anirudh Gupta

March 2021

1 Objective

The objective of this project is to estimate the traffic density at various times from a video clip of traffic light camera. Below we have described various tasks we have performed in completing this project and further improving it.

2 Subtask1

OpenCV algorithms work best on rectangular frames. But when a camera is deployed on a real road, it is very hard to position the camera so that the road to be monitored is a perfect rectangle, and the camera gets a top view of that rectangle. Typically the road to be monitored is at an arbitrary angle to the camera, and many additional objects are part of the frame. For useful tasks like traffic density estimation on a particular road stretch, it is therefore important to first correct the camera angle in software, and crop the image to remove the additional objects. This is the goal of the Subtask1.

This code takes the input image and asks the user the corner points of the road on which you want to estimate traffic density. The code then outputs the Top view of the image as well as its cropped version.

3 Subtask2

This code takes a video and a background image of an empty road (same as in the video) as input. It outputs queue density and dynamic density for each video frame. It also saves the data in a csv file. Queue density is density of all vehicles queued (either standing or moving). Dynamic density is the density of those vehicles which are not standing but moving in that same stretch. This code uses Subtask1 to crop the top view of the road in each video frame for better accuracy.

4 Optimization

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 analysis.

In addition to metrics, we have to understand what baseline and our methods of optimization are. Baseline is the method against which we compare other methods/parameters, and see whether we are getting a better or worse trade-off. The output in subtask2 acts as the baseline for the analysis of each method.

Next we will look at various optimization methods with a trade-off between error(utility) and time. The analysis of same is provided the analysis folder where we have taken a certain video of size 6 minutes from Lajpatnagar junction, Delhi.

- **Method1:** sub-sampling frames – processing every x frame i.e. process frame N and then frame N+x, and for all intermediate frames just use the value obtained for N - total processing time will reduce, but utility might decrease as intermediate frames values might differ from baseline. Parameter for this method is x.
- **Method2:** reduce resolution for each frame. Lower resolution frames might be processed faster, but having higher errors. Parameter is resolution XxY.
- **Method3:** split work spatially across threads (application level pthreads) by giving each thread part of a frame to process. Parameter can be number of splits i.e. number of threads, if each thread gets one split.
- **Method4:** split work temporally across threads (application level pthreads), by giving consecutive frames to different threads for processing. Parameter is number of threads.

- **Method1-static:** Here method1 and method3 are combined for further optimization and analysis.
- **Method2-static:** Here method2 and method3 are combined for further optimization and analysis.