Velocity Estimation in Real Time Videos

Shaddy Garg
1611061
Dept. of Electronics and Communication
Indian Institute of Technology, Roorkee
Roorkee, India
sgarg1@ec.iitr.ac.in

Surya Saini
1611068
Dept. of Electronics and Communication
Indian Institute of Technology, Roorkee
Roorkee, India
ssaini@ec.iitr.ac.in

Swapnil Negi
1611071
Dept. of Electronics and Communication
Indian Institute of Technology, Roorkee
Roorkee, India
snegi@ec.iitr.ac.in

***Abstract*—Velocity estimation from real time videos is an important problem in the field of image processing and machine vision. Monitoring real-time traffic, the consciousness for the self-driving vehicles are a few examples.**
In this paper we present a novel method of estimating real time object velocity using optical flow and object tracking to estimate pixel displacement data which gives us the average velocity of the tracked object in pixels per second (PPS).

***Index Terms*—optical flow, object tracking, video processing, OpenCV, Lucas Kanade Algorithm**

I. Introduction

Velocity estimation from real time videos has a long history in computer vision. Fruitful approaches have relied on background removal, object tracking, and optical flow. With the recent boom in fields of deep learning and availability of extensive computational power, Neural Network based velocity estimators provide state of the art results. These Neural Network rely on object detection in individual frame and estimating the velocity from the correspondence with the next frames. While these methods have enjoyed great success, they rely heavily on computational power and depending upon the complexity of the network, the time taken to estimate the velocity varies. Sometimes the network is so complex that it takes a long time to perform the required task, no longer making the approach useful in a real case scenario.
In our method we estimate the velocity of the objects by applying optical flow over the incoming video frames, the computational speed of the algorithm is reduced by selecting the region of interest in which we start tracking our object. Specifying the region of interest also helps in improving the overall accuracy, as now optical flow is applied only in the region of interest and our results are not affected by unwanted background movements.

II. Methodology

We combine the principles of object detection and optical flow to track the moving objects and then correspondingly find the velocities for the same.

A. Object Detection

The aim of our algorithm is to track the movement of the objects and estimate their velocity. So the first task that we perform is to find the object, we need to track in our video. One of the proposed and well-used method is background subtraction. In background subtraction generally, the first frame of the video(under the assumption that the camera taking the video is stationary) is removed from each subsequent frame. This renders a black image and any anomaly(motion) in the subsequent frames are rendered as white coloured blobs on a black background. We define a region of interest manually

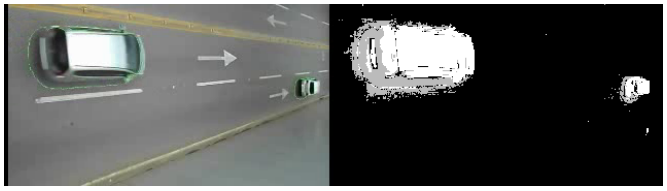

Fig. 1. Background removal using KNN background removal algorithm

based on the camera specifications which records the video and the area which is of interest to us. In this region of interest, we detect motion by finding if a white pixel occurs repeatedly for more than a threshold number of time frames. After this we apply optical flow on the original frame in which we select features that may be of interest using OpenCV's *goodFeaturesToTrack*, which determines the strong corners in the region of interest of the video frame.

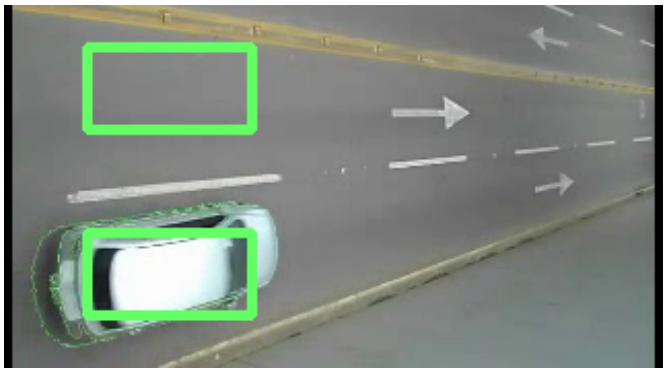

Fig. 2. Manually defining the Region of interest

B. Optical Flow

After the object is first detected, the task of tracking the object starts. To do this, we have implemented optical flow

over the object inside the defined region of interest. We use the *Lucas Kanade* algorithm for optical flow estimation which assumes that the flow is essentially constant in the local neighbourhood of the pixel under consideration, and then solves the basic optical flow equations for all the pixels in that neighbourhood, using the least squares criterion. Optical flow helps in tracking the object by estimating the one to one correspondence between multiple points (that are being used for tracking) in different frames.

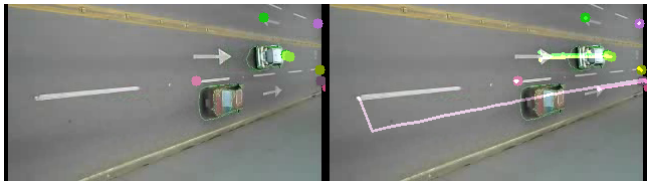

Fig. 3. Lukas Kanade optical flow

To automate the process of optical flow once the object to be tracked is detected, a trigger is applied to the run the optical flow on the object. Once we find a pixel whose value is greater than a threshold value in our defined region of interest, this trigger is provided using the background subtracted frame. Based on the assumption that camera is stationary, we infer that existence of a non-black pixel in the region of interest is a moving object that is to be tracked.

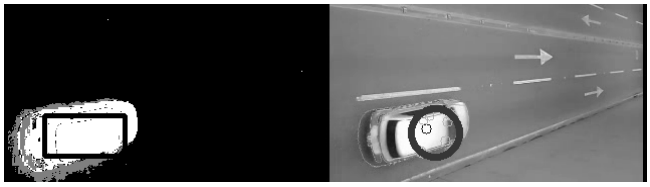

Fig. 4. Triggering tracking using detection of white pixel

C. Velocity Estimation

The next step in the method is to actually estimate the velocity. To do this, we use the data obtained in the previous step (i.e. Optical Flow). Optical Flow provides us with information about the new positions of the points that are being tracked. We then use these positions to calculate the centroid of the object being tracked. After that, we find the relative displacements between existing centroid and the new centroid using the Euclidean distance formula. We divide this by the sampling rate of the video (50 ms in our case) to calculate the velocity in pixels per second for the current iteration. Lastly, we take take the average value of all the computed velocity to obtain the average speed of the moving vehicle.

III. Results

A. Implementation Details

We set the parameters for the *Lucas Kanade Optical FLow Algorithm* defining the window size, max-level, number of points to track as:

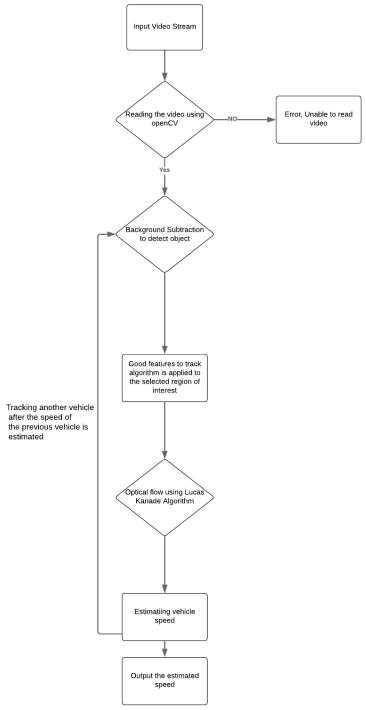

Fig. 5. Actual implementation of our algorithm

```
# Parameters for Lucas Kanade optical flow
lk_params = dict( winSize = (15,15),
                  maxLevel = 2,
                  criteria =
                      (cv2.TERM_CRITERIA_EPS |
                      cv2.TERM_CRITERIA_COUNT, 10,
                      0.03))
```

The background subtraction is then implemented using the openCV's *createBackgroundSubtractorKNN*. The *createBackgroundSubtractorKNN* is preferred over *createBackgroundSubtractorMOG2* because it provides less distortion in the background subtracted image when a car passes by. We define our region of interest based on camera calibrations use our background subtracted video to detect any motion inside the region of interest. We also set 10 good features to track inside this ROI. We set a threshold of 125 in our region of interest. If we detect the value at the center of the region of interest to be greater than the threshold continuously for 4 frames, we trigger object tracking for that area of interest.

After the trigger, these 10 points are continuously tracked till they reach the end of the frame or for a set number of frames(25). This algorithm continuously provides us the values

of the updated locations of these 10 points. We use these positions to find the minimum enclosing circle for these points and draw the circle on the frame for better human perception. The actual tracking of the moving object is done through the *Pyramidal Lucas Kanade Algorithm*. We pass the newly greyscale converted frame and the old one. This returns the location of those 10 points which are being tracked.

```
new_corners, st, err =
    cv2.calcOpticalFlowPyrLK(oldFrameGray,
    frameGray, old_corners, None, **lk_params)
```

An essential feature of our implementation is pruning away of the far away points. Not all *good to track points* flow with the moving object, so to reduce the error caused by the non-flowing points we set a threshold on the absolute distance of that point from the centroid of the points. If the absolute distance is greater than 30 pixels (threshold value), then we no longer use the point to calculate the centroid in the next frame. To calculate the speed of the tracked object, we find the absolute difference between these centroids. Total numbers of frames are calculated using a counter variable. The calculated optical displacement is multiplied by 20 because we sample our video frames after 50 ms. This gives us the result in pixels per second.

B. Implementation Results

We tested our implementation on videos from different sources. The results are as follows:

Fig. 6. Tracking of a car for a sample video

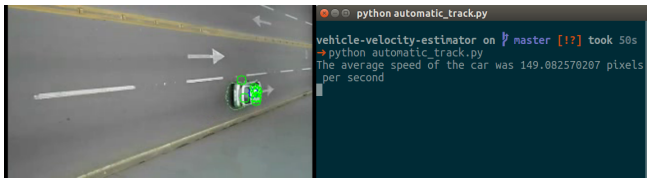

Fig. 7. The velocity of the car being tracked

Fig. 8. Tracking of a car for another sample video

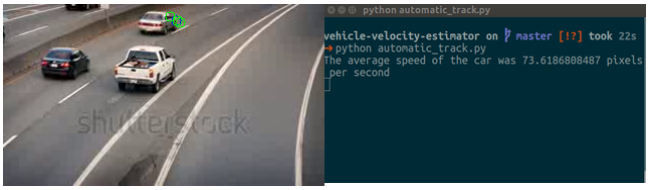

Fig. 9. The velocity of the car being tracked

```
vehicle-velocity-estimator on master [!?] took 20s
→ python automatic_track.py
The average speed of the car was 149.082570207 pixels per second
The average speed of the car was 152.947870229 pixels per second
The average speed of the car was 155.85483707 pixels per second
The average speed of the car was 153.529758457 pixels per second
The average speed of the car was 174.202859739 pixels per second
The average speed of the car was 153.086171069 pixels per second
The average speed of the car was 154.889842691 pixels per second
The average speed of the car was 149.613518069 pixels per second
```

Fig. 10. The final output of a video

C. Generalizing to videos from different Sources

The region of interest is specific to a video source and needs to be calibrated accordingly for different sources of video. Our implementation provides 2-lane vehicle tracking where the region of interest is set based upon the lane position. A generalization of the video can be implemented using processing over different threads, where one thread runs the code to detect the object using background subtraction. Once the object is detected it sends a trigger to a parallel running code in a different thread that tracks an object using optical flow. Based on the center of the detected object from background subtracted video, an automated region of interest can be selected.

D. Limitations

- We manually have to select the region of interest based on the camera that provides the video.
- Current implementation can only track one car at a time, in either of the lanes.
- Our implementation outputs the estimated speed in pixels per second and not in a standard SI unit. This is due to lack of Real world information about the camera distance from the objects being tracked, the focal length of the camera, and the span of region that is being covered by the object.

E. Future Prospects of the Implementation

- Automating the selection of region of interest: This can be done by using object detection algorithms for detecting vehicles in frames. Presently our implementation is generalized to detect any kind of motion. It needs to be trained to be specific to the task of detecting cars. A possible implementation can use HoG Filters to detect cars and estimate the size of the detected car around which our region of interest can be drawn.
- Our implementation needs to improved to track multiple vehicles together. This may be achieved using parallel

processing making the entire algorithm computationally more intensive.

- A mapping of image to real world coordinates need to be included to output the speed in SI units.

ACKNOWLEDGMENT

We would like to thank Professor Saumik Bhattacharya for giving us this golden opportunity to work on image processing algorithms as well as for his continuous support and guidance throughout the duration of the course project.

REFERENCES

[1] Optical Flow Documentation: Python OpenCV
[2] Documentation of background subtraction using KNN: Python OpenCV
[3] Velocity Detection in Video Frames: GeeksForGeeks
[4] Moving Vehicle Detection and Speed Measurement in Video Sequence Ms. Bhagyashri Makwana and Prof. Pravesh Kumar Goel