Traffic Surveillance

Vehicle Tracking

Kumar Abhinav

Indian Institute of Technology, Kharagpur

Summer Research Project

Under the guidance of

Prof. Lam Siew Kei

PhD. Kratika Garg

School of Computer Science and Engineering

Nanyang Technological University, Singapore

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my guide **Kratika Garg** for her exemplary guidance, monitoring and constant encouragement throughout the course of this project. The help and guidance given by her time to time shall carry me a long way in the field of Computer Vision.

I also take this opportunity to express a deep sense of gratitude to **Lam Siew Ki** for offering me the opportunity to work for summer project at School of Computer Science and Engineering, NTU and for his cordial support and guidance, which helped me in completing this internship.

I am obliged to staff members of Hardware and Embedded Systems Lab, for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Table of Contents

Introduction

A vision-based detecting and tracking vehicle in video streams is an important research in computer vision, and it plays an important role in ITS. The aim of motion detection is to get the changed region from the background image in video sequences, and it is very important for targets classification and tracking motion objects.

The vehicle tracks, or trajectories, are measured over a length of roadway, rather than at a single point, thus it is possible to measure true density instead of simply recording detector occupancy. The additional information from the vehicle trajectories could lead to improved incident detection, both by detecting stopped vehicles within the camera's field of view and by identifying lane change maneuvers or acceleration/deceleration patterns that are indicative of incidents beyond the camera's field of view. The trajectory data could be used to automate previously labor intensive trajectory studies, such as examining vehicle maneuvers in weaving sections or bottlenecks.

The representative works in vision based vehicle tracking are:

- Background subtraction and Mean shift algorithm
- Kalman Filter based tracking
- SVM based particle Filtering
- Feature based Optical Flow
- Spatio-temporal MRF

The various methods proposed above have their own advantages and some even track vehicles accurately. However existing techniques based on vehicle counting and tracking suffer from low accuracy due to sensitivity to illumination changes, occlusions, congestions etc. In addition, existing holistic-based methods cannot be implemented in real-time due to high computational complexity. The current work on vehicle tracking [1] is a block based holistic approach and employs variance as a means for detecting the occupancy of vehicles on pre-defined blocks. The tracking is done based on the occupancy data of the block.

Literature Review

The current research in Vehicle tracking follows following methods:

- Moving Vehicle Extraction
- Region Based Vehicle Tracking
- Vehicle classification (Optional)

Moving Vehicle Extraction, mainly consists of two major processes namely, i) Background subtraction and ii) Shadow removal. This process can have problem while implementing in real time due to high computational complexity. Also this method can have low accuracy due to sensitivity to illumination changes, occlusions, congestions etc.

Region Based Vehicle Tracking, a lot of various methods has been developed till date. Most of these methods predicts the future position of the vehicle based on state representation, e.g. Kalman filter .The comparison of various Vehicle Tracking methods can be found out in the review paper [2].

The current work in Vehicle Tracking uses the method from paper [1] for Moving vehicle extraction, which has low computational complexity and is robust. The work for Vehicle tracking is inspired from Graph based mapping [3] and is completely new in its approach.

Framework

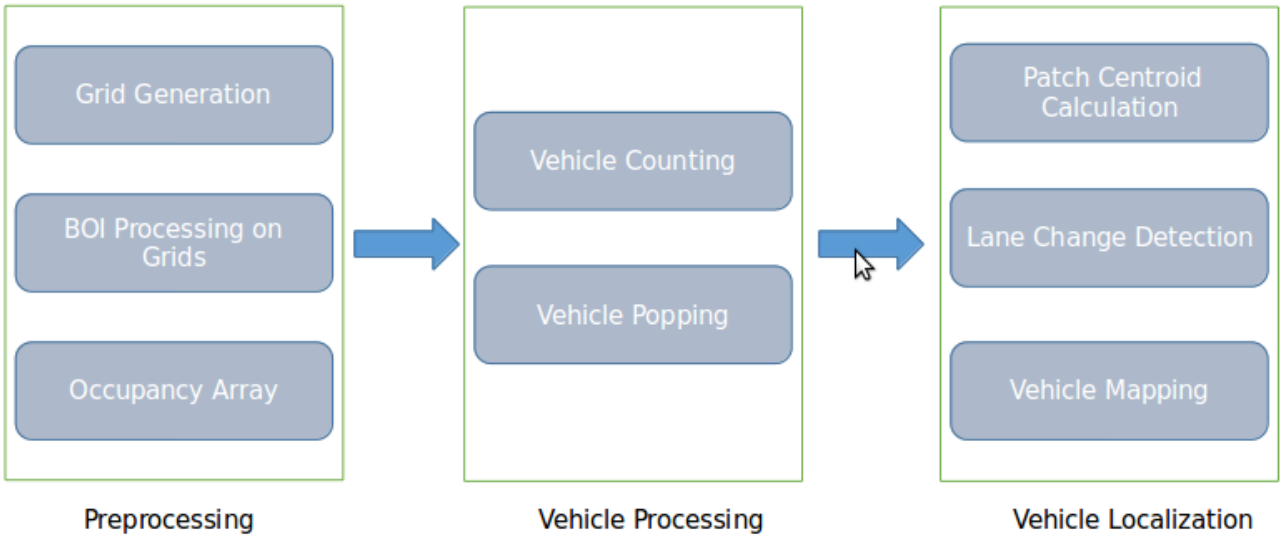


Figure 1: Tracking Framework

The proposed system mainly consists of three steps including Preprocessing, Vehicle processing and Vehicle Localization.

The **Preprocessing** step is Vehicle region extraction based on method [1] employing variance as a means for detecting the occupancy of vehicles on pre-defined region of interest and has comparable accuracy with existing high complexity holistic methods.

The **Vehicle Processing** step updates the vehicle count by adding the new vehicle at start and removing the vehicle at the end.

The **Vehicle Localization** step accounts for updating the vehicle position by taking into account Lane change and mapping with the centroid of the patch.

Preprocessing

Grid generation

- Lanes were generated based on the manual input of user for the first time.
- Lanes were then divided into subLanes, each lane divided into 3 subLanes.
- Each subLane is then divided into fixed number of vertical blocks so as to form a grid.
- Grid is finally represented as a 2-D array.

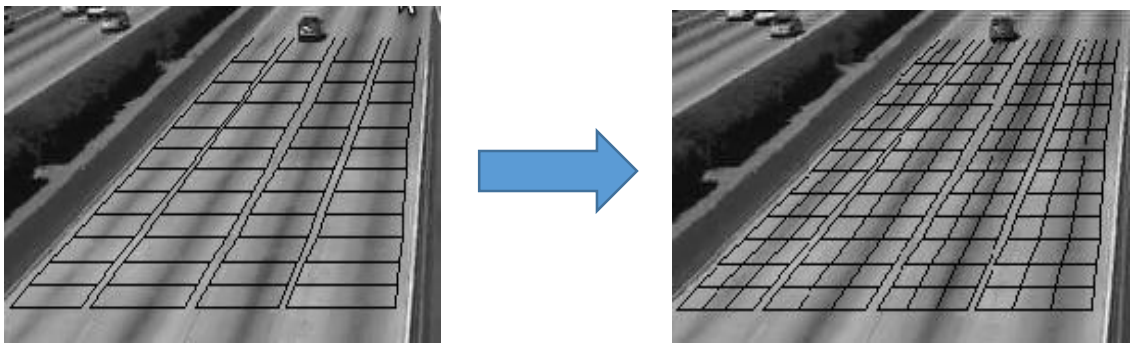


Figure 2: subLane generator

BOI processing on grids

This process is based on the method described in paper [1]. Variance was used as a means for detecting the occupancy of vehicles on pre-defined region of interest and as result that particular grid is colored white.

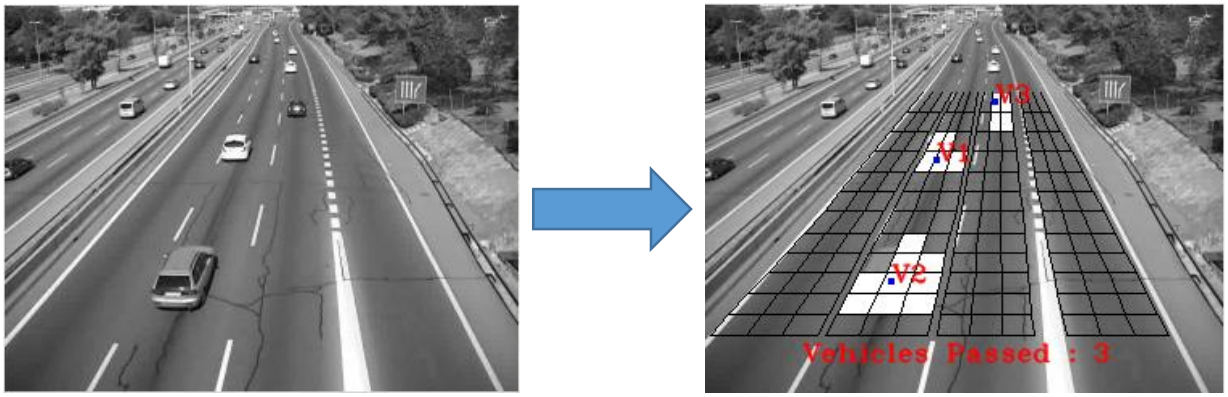


Figure 3: BOI processing

Occupancy array

The 2-D array was represented as a combination of 0's and 1's based on occupancy of that grid. '1' is marked for white colored and '0' for no color.

If at least "2" sublanes in the lane is colored then the isLaneColored at that index is "1".

Example:

Figure 4: Preprocessing step

Terminology:

- Occupancy array :
  isGridColored[2][h][i] , **isLaneColored**[2][h][i] , [0]-previous frame
  [1]-current frame
- Array of vector to store pair of Vehicle no. and Index no. at particular lane :
  Track[h]{ pair< Vehicle_no , Index> } , h : Lane number
- Mapping vehicle number to set(vector) of positions :
  Position : { int } -----> { vector<int> }
- Array of centroids of patches :
  patchCentroid[h]{pair < sublane , index > } , h : Lane number

Vehicle Processing

Vehicle counting

It is used for adding a new vehicle entering the lane. It contains a set of entry check so as to confirm the entry of a new vehicle. The entry check is maintained so as to compensate for the fluctuation in vehicle region detection which can result in the false counting of vehicle.

Algorithm:

1. For frame 1 :
    a. For each lane , iterate through the index from top to bottom*
    b. Get the set of white patches for each lane
    c. Update the vehicle counter for each set of patches
    d. Associate the lowest index of each patch as position of respective vehicle

    *Top to bottom was used as farthest patch should be ranked first

2. For next set of frames :
    a. For each lane , iterate through index , *i*=0 to max_index
    b. Check if at index,*i* lane is colored in current frame and not in previous
    c. Check if that is empty :
        i. If Yes: Update the vehicle counter and give position of that vehicle as 'i'.
        ii. If No: Update vehicle counter and position only when the new vehicle position(i) is at some distance from earlier existing vehicle

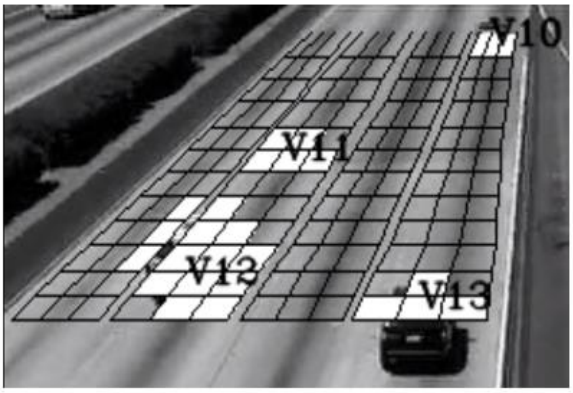
After the implementation of following algorithm, all vehicles entering the lane are initialized into its respective **Track** array.

Vehicle Popping

This process removes the vehicle from **Track** which are leaving the lane or not detected due to simultaneous non detection.

Algorithm:

1. Case 1 : Vehicle actually leaving the lane
    a. For each lane , for *i* = max_index - 1
    b. Check if lane isn't colored for current and previous frame at index, *i* and *i-1* respectively and was colored at previous frame at index *i*
        i. If Yes : Pop out the vehicle
2. Case 2 : Simultaneous non detection
    a. For each lane , get the vehicle id of front vehicle in **Track**
    b. Check the last three position values of that vehicle (**Position[vID]**)
    c. If all three values = -1 , pop out the vehicle



Track[0] : { }

Track[1] : { < 11 , 7 > , < 12 , 0 > }

Track[2] : { }

Track[3] : { < 10 , 12 > , < 13 , 0 >}

Figure 5: Vehicle processing step

Vehicle Localization

Patch Centroid Calculation

The entire image frame has been converted to a 2-D array containing the occupancy data. The vehicle presence forms a set of white connected component patch in the image which in terms of 2-D array is the connected component of set of “1’s”. Thus calculation of the centroid of each patch results in the position of all the vehicles in the image frame.

Few important points for this method are:

- To initiate patch transversal (a) there should be two neighboring grids colored or (b) for last sublane, intiated with only one grid colored
- Four connected Breadth First Algorithm (BFS) was used
- Wing span / subLane span of “2” blocks was allowed from central node. This was used to separate two side by side vehicle
- Diagonal vehicle occlusion handled when vertical span of part of patch in next lane exceeds certain range
- Centroid is calculated from arithmetic mean of all subpatch’s coordinate value
- In the same lane two centroid points shouldn’t be generated close enough

Algorithm:

1. For each sublane, iterate through all index , *i* = 0 to max_index
2. Initiate the patch transversal if the earlier above condition in achieved
3. Follow the above mentioned points for patch transversal and calculate the patch centroid for the same
4. Push each centroid points in descending order of index in **patchCentroid**

Lane Change Detection

Lane change is tracking is an essential part of vehicle tracking. Using the proposed algorithm the lane change was smoothly detected provided that the Preprocessing step is perfectly done. This step is executed before Vehicle mapping so as to have actual amount of vehicle in the lane.

This step compares two set of data notably:

- **patchCentroid** : current frame value (updated)
- **Track** : previous frame value

Algorithm:

1. For each lane , iterate through all the patches in **patchCentroid[h]**
2. Check if there is any vehicle in **Track[h] ,** whose index is close to patch's index
    a. If Yes : No lane change
    b. If No :
    c. Check for each vehicle in Left of current lane, is its index is close to patch's index
        i. If Yes : Left lane change // save the vehicle information as V1
        ii. If No : No left lane change
    d. Check for each vehicle in Right of current lane, is its index is close to patch's index
        i. If Yes : Right lane change // save the vehicle information as V2
        ii. If No : No right lane change
    e. If there's Left lane change , Check if there's a patch in **patchCentroid[left]** whose index is close to V1 :
        i. If Yes : No left lane change
        ii. If No : Final left lane change , remove V1 from Track[left] and insert it in Track[h]
    f. If there's Right lane change , Check if there's a patch in **patchCentroid[right]** whose index is close to V2 :
        i. If Yes : No right lane change

ii. If No : Final right lane change , remove V2 from Track[right] and insert it in Track[h]

Vehicle Mapping

This is the final and important step of vehicle tracking. Before this step we have count of all vehicles in the image frame and set of all patches, this step links the patch to its correct vehicle entity. Thus this step is being termed as the Vehicle updating step.

The process is termed as condition based mapping and is inspired from Graph based vehicle mapping [3].

Possible cases while vehicle mapping:

1. Case 1 : Occlusion
2. Case 2 : Non detection of earlier detected vehicle
3. Case 3 : Normal one to one mapping
4. Case 4 : Detection of earlier non-detected vehicle in middle of path

For each lane based on comparison of **Track.size()** and **patchCenroid.size()** we have following cases:

1. Condition 1 : **Track.size() = patchCentroid.size()** // No. of vehicles equals no. of centroids
2. Condition 2 : **Track.size() > patchCentroid.size()** // No. of vehicles is more than no. of centroids
3. Condition 3 : **Track.size() < patchCentroid.size()** // No. of vehicles is less than no. of centroids

Algorithm* :

1. If Condition 1 :
    a. Case 3
    b. Each vehicle is mapped with centroid respectively
2. If Condition 2 :
    a. If **patchCentroid** is empty , for all vehicle in **Track**, push its corresponding position as
    "-1" in the **Position** vector
    b. Iterate through vehicles in **Track** in that lane , check if vehicle.index > patch.index
        i. If Yes : Check if two neighboring vehicles are close enough
            - If Yes : Case 1 , mark both the vehicles to same patch
            - If No : Case 2 , push the corresponding position as "-1"
        ii. If No : Case 3
3. If Condition 3 :
    a. Increase the vehicle counter and mark the patch which is new generated
    b. Link the new generated vehicle to the marked patch
    c. Map all else vehicle as in Case 3

*For all the update statements, a check is done so as to ensure that the vehicle position isn't getting fluctuated by a large value in between two frames.

After the execution of this step, we get the final vehicle tracking model as desired, with robust tracking and low computational cost as compared to previous trackers.

Bidirectional Tracking

For a stationary camera, there could be two directions of the traffic possible, flowing towards the camera and flowing away. Here are the improvements done in bidirectional tracking:

- Traffic flow direction is calculated for each lane (1 : away , -1 : towards)
- Traffic flow direction(TFD) was calculated based on entry and exit index of each lane
- For TFD = -1 , the 2-D array for that particular lane is inverted so as to maintain the coordinate system

Figure 6: Bidirectional

Discussion

- New vehicle generation was first accounted in *Vehicle Counting* but later shifted to *Vehicle Localization,* as the method of vehicle counting should be as simple as possible as it is the backbone of Tracking.
- Last 3 frame values was used for *Vehicle Popping* , as it was found optimum but can be change for later reference
- For all references of “if two vehicles are near” refers to the condition that the index of two vehicles are less than “*3*”.
- For Bidirectional tracking, threshold parameters can be changed as the vertical span of vehicle vary with camera perspective.

Future Work

- Moving vehicle shadow removal for block occupancy detection, this hampers the tracking process.
- Adaptive thresholding in order to have better preprocessing.
- Camera perspective to be added for grid generation.
- Vehicle counting condition check to be robust.

References

[1] Kratika Garg, Siew-Kei Lam and Thambipillai Srikanthan - Real-time Road Traffic Density Estimation using Block Variance

[2] B. Tian *et al*., "Hierarchical and Networked Vehicle Surveillance in ITS: A Survey,"

[3] Z. f. Liu and Z. You, "A Real-time Vision-based Vehicle Tracking and Traffic Surveillance “

[4] J. C. Lai, S. S. Huang and C. C. Tseng, "Image-based vehicle tracking and classification on the Highway “