ROAD INTERSECTION MONITORING FROM VIDEO WITH LARGE PERSPECTIVE DEFORMATION

Takashi Furuya[1], Camillo J. Taylor[2]

Computer and Information Science, University of Pennsylvania

[1] Tel: 215-450-2133, Email: tfu@seas.upenn.edu

[2] 474 Levine Hall (GRW) 3330 Walnut Street, Philadelphia, PA 19104-6389

Tel: 215-898-0376, Fax: 215-573-2048, Email: cjtaylor@cis.upenn.edu

June 7, 2014

ABSTRACT

We propose a method to extract traffic information from videos of road intersections, taken from a low-resolution camera fixed 2 to 7m above ground. First, feature points are detected and tracked with the Kanade-Lucas-Tomasi tracker to obtain trajectories. These trajectories are then segmented into groups corresponding to vehicles so that the vehicle's path, velocity, time of entry and exit can be determined.

Keywords: Vehicle Detection, Intersection Monitoring, Feature Tracking, Background Subtraction

INTRODUCTION

Traffic flow records are essential for both urban planning and traffic management. These could be collected from inexpensive cameras which are already installed in highways and road intersections; however, extracting useful information from these videos requires significant manual labor or the use of commercial software.

As an example, traffic studies in Philadelphia are often conducted by temporarily deploying a video camera at an intersection. Footage acquired from this camera is used to analyze routes taken by cars over the recorded period. Once the video has been obtained, a human operator may manually annotate vehicle turns and counts. This process is time consuming and labor intensive. An alternative is to rely on commercial entities such as Miovision Technologies [1] which has an automated/semi-automated system to accomplish such tasks for a fee.

As part of the U.S. DOT funded T-SET [2] program, a project was initiated to develop an open source video analysis software to reduce the costs of extracting traffic information from videos. This would make it possible to gather more traffic flow information in a timely manner. This paper presents part of this effort. Source code and documentation for the system are currently available in a public repository [3].

Figure 1: A sample snapshot from video.

To give an idea for the type of data we are working with, Figure 1 shows a single frame from a video provided by the Delaware Valley Regional Planning Commission (DVRPC).

Three major factors that make the above video difficult to analyze are listed below:

- Large perspective deformation

  Unlike a nadir view such as would be provided by satellites or helicopters, the relatively low position of the camera causes closer objects to appear larger and move faster than those further away from the camera. Further, the same vehicle does not always maintain a fixed appearance because it may make turns that may reveal or occlude various aspects. Our tracking system needs to account for these effects.

- Shadows

  The shadows associated with the vehicles can confound the tracking process. It is often difficult to discriminate between motions due to a vehicle and motions due to a vehicles shadow. This can lead to incorrect counts.

- Diverse motion

  The vehicles in this video may stop at a variety of positions and make a variety of turns. We can contrast this with the problem of analyzing video of highway traffic where the flows are more constrained and predictable.

Our proposed algorithm aims to tackle these three issues by combining several techniques, which are described in the next section.

PREVIOUS RESEARCH

Various works on video-based traffic monitoring has been published since the early 1990s. Many of these use a variant or a combination of the following strategies [4]:

- Region-based tracking

  These methods begin with a background subtraction step which is used to identify regions in the image corresponding to vehicles. Image regions are typically delineated using a connected components procedure and the resulting blobs are individually tracked over time.

  Typically an adaptive Gaussian mixture model is used for background subtraction [5]. It creates a background model that evolves over time to account for gradual lighting changes.

  Once the blobs are obtained, they need to be tracked. Inter-frame blob matching can be accomplished by comparing all of the blobs' regions from the first frame with those from the second. This has been done by choosing pairs with a large area of overlap [6] or with the highest cross correlation [4]. Kalman filtering is generally done as a final step [4, 6, 7].

  A major drawback with this approach is that it is not robust to the presence of shadows and congestion; nearby blobs are often connected to each other as shown in Figure 2a. On the contrary, distant objects may not be visible or appear partially fragmented, as in Figure 2b. Their small size and slow motion on screen make them susceptible to noise.

- Model-based tracking

  Unlike the previous methodology, model-based tracking relies on additional predefined models such as the vehicle's dimensions, appearance, and path [8, 9]. Our goal in this effort was to make the system as flexible as possible and to reduce the burden on the user so methods that relied on on prior estimates or detailed models were not considered.

- Optical-flow based tracking

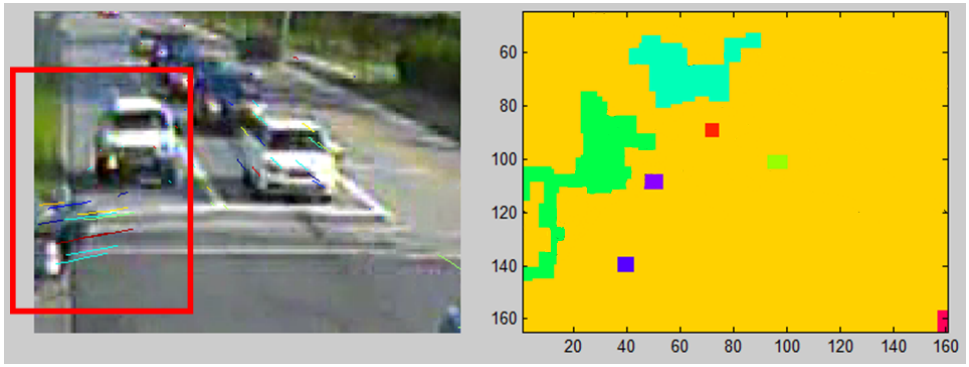
  Optical flow computes the instantaneous velocity between two frames for a selected subset of pixels. Lucas and Kanade's pyramid based algorithm is commonly used. The resulting flow estimates must still be analyzed and segmented.

- Feature-based tracking

  In this approach, distinguishing features points, such as Harris corners, are detected and tracked over time. The trajectories formed by these points are grouped such that each group corresponds to a path taken by a single vehicle. This is generally done in a two-step process: first, an undirected graph is constructed, where nodes represent trajectories and edges represent similarities between pairs of trajectories. Then the graph is used to group the nodes.

  Similarity between trajectories may be computed by checking the maximum distance between them. Slight variations of this approach have been studied in tracking vehicles at intersections and also people in crowded environments [10, 11].

(a) A frame (left) and its binary foreground image (right). Note that the red car's shadow connects with the blue car in front.



(b) A zoomed in region from another frame with its foreground's connected components displayed in different colors. It is difficult to capture cars turning in the region specified by the rectangle with a red outline.

Figure 2: Background subtraction failing.

Our proposed method builds on this last technique. In addition our work takes into account the perspective deformations induced by the cameras vantage point.

METHOD

Our system is divided into three main steps: (1) determining the projective transformation; (2) extracting trajectories; and (3) grouping them. The entire process is summarized in Figure 3.

Projective Transformation Determination

This calibration step is performed once for every video based on input from the user. More specifically the user is asked to identify two sets of parallel lines in the frame as shown in Figure 4. These usually correspond to the two roadways that are meeting. The system then computes vanishing points for both sets of lines and uses these results to compute a rectifying homography, T, which can be used to map points on the image to their location in a top down view of the scene. This transformation allows us to estimate how the feature points are actually moving through the scene.

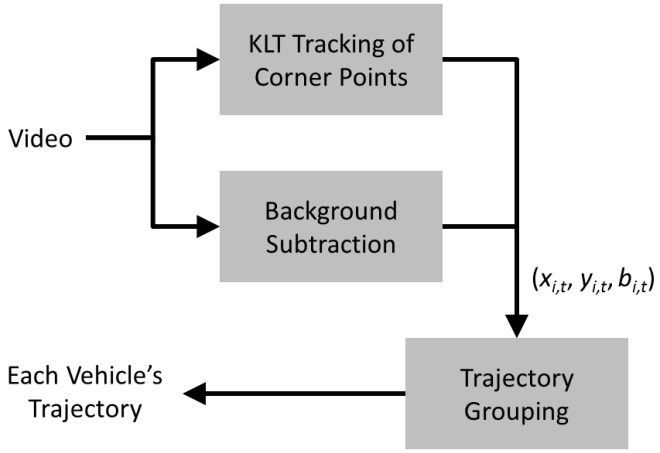


Figure 3: Block diagram of proposed algorithm.

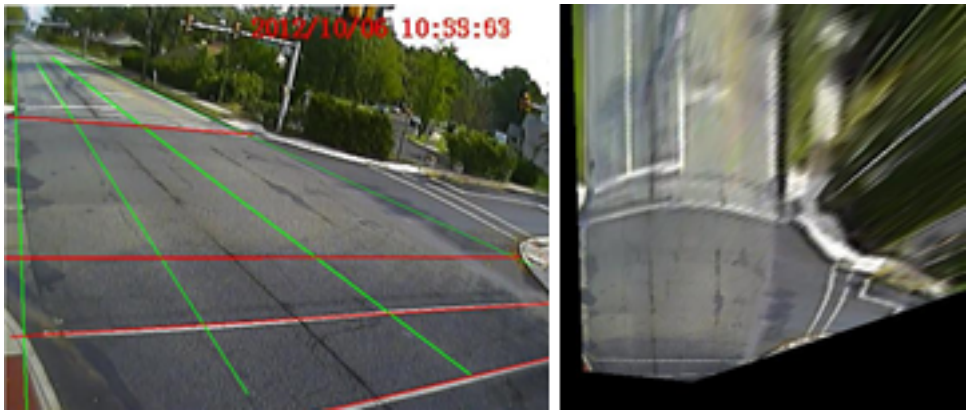


Figure 4: Lines drawn by user (left) and effect of applying the rectifying transformation T to the original image (right).

Trajectory Extraction

The second step processes the video and outputs a list of trajectories. Initially, good features to track are detected in the first frame [12]. They are tracked over time using the Kanade-Lucas-Tomasi tracker. We used the OpenCV implementation for the corner detector and tracker. Figure 4 shows some of these points and their trajectories.

In order to detect the corners, we first convert the color frame image to gray scale using the following formula to calculate the luma (brightness):

$$Y = 0.299R + 0.587G + 0.114B$$

where R, G, and B are each of the color channel values and Y is the gray scale output.

We determined that sampling up to 400 corners in each frame is sufficient to detect points on most vehicles. We also set the minimum Euclidean distance between the points to be 5 pixels and the block size (used for calculating the derivative covariance matrix for each pixel neighborhood) to be 3. New points are sampled every 15 frames as the tracker

Figure 5: Feature points and their trajectories.

loses points over time. We found this rate to be appropriate for our video. The tracking parameters are summarized in Table 1.

We incorporate a forward-backward error threshold, or a maximum bidirectional error as is done in the Matlab implementation of the KLT tracker [13]. This error measure is calculated by tracking a point in frame i to frame $i+1$ and then tracking that point in frame $i+1$ back to frame i. The Euclidean distance between the original and the new point both in frame i is computed. If the distance is larger than the value we set, we mark that point as invalid. We found that this significantly improves the quality of our trajectories. The resulting tracking procedure yields the position of each feature in every frame, $(x_{i,t}, y_{i,t})$, along with an estimate for its instantaneous velocity, $(\dot{x}_{i,t}, \dot{y}_{i,t})$.

Table 1: Parameters used for KLT tracker.

Parameter Name	Value
Block size (size of neighborhood search window)	31 x 31
Maximum number of pyramid levels	3
Maximum number of iterations	30
Maximum bidirectional error	2

We also perform a background subtraction procedure to help identify foreground objects in the frame. We used a uni-modal Gaussian background model in RGB space which was constructed by sampling 1000 random frames from our hour long video. As a post-processing step, morphological operations are applied to the binary foreground image to remove noise. This is done by opening then closing with a circular structuring element of size 3 x 3 pixels. This procedure contributes a fifth value to each tracked feature, $b_{i,t}$, which is 1 if the feature is currently considered part of the background, and 0 otherwise.

Figure 6: Vehicle trajectories after feature point trajectories are grouped.

To prevent tracking stationary points in the background, we stop tracking if the following two conditions are met for 4 consecutive frames:

$$b_{i,t} = 1 \text{ and } \dot{x}_{i,t}^2 + \dot{y}_{i,t}^2 < 1.5^2$$

The two conditions are necessary since $b_{i,t}$ by itself is noisy and thus unreliable. Velocity by itself is also insufficient since the vehicles may sometimes be stationary. Finally, short trajectories (i.e. those spanning less than 4 frames in our case) are also removed since they do not form a reliable trajectory and are more difficult to group correctly.

Trajectory Grouping

The last step is to group the acquired trajectories so that each group corresponds to a vehicle. Figure 5 shows the results of this grouping stage using distinct colors and Figure 6 shows the resultant vehicle trajectories.

The trajectories from the previous section are first transformed into the top-down view using the rectifying homography T obtained in the first section. Then, their velocities (in the rectified space) are computed. Trajectories containing points which fall outside the region of interest after transformation are removed. The system can then compute a measure of similarity between any two features, i and j, based on these position and velocity estimates as follows:

$$d_{i,j,t}^2 = (x_{i,t} - x_{j,t})^2 + (y_{i,t} - y_{j,t})^2$$
$$s_{i,j,t}^2 = (\dot{x}_{i,t} - \dot{x}_{j,t})^2 + (\dot{y}_{i,t} - \dot{y}_{j,t})^2$$

Two trajectories are deemed to belong to the same object if they satisfy the following condition,

$$\frac{\sum_t S_{i,j,t}}{N} > 0.9$$

Where N is the total number of frames both trajectories are present, and $S_{i,j,t}$ is a binary value indicating their similarity at every point in time, defined as,

$$S_{i,j,t} = \begin{cases} 1, & \text{if } d_{i,j,t} < 26 \text{ and } s_{i,j,t} < 3.6 \\ 0, & \text{otherwise} \end{cases}$$

This means that the two trajectories belong to the same object if their distance remains small and velocities are similar for 90% of the time.

A graph can be constructed, where every trajectory is a node and an edge connecting them represents the fact that they are thought to belong to the same object. It is possible to extract each vehicle's group of trajectories by applying Dulmage-Mendelsohn decomposition on this graph's adjacency matrix. Groups with less than 4 trajectories are removed since they are not reliable.

The trajectory of each vehicle is calculated by averaging all of the feature point trajectories present at each frame. Then, the final merged trajectory is smoothed using a centered weighted moving average as follows,

$$\bar{x_t} = \frac{\sum_{i=t-k}^{t+k} n_i x_i}{\sum_{i=t-k}^{t+k} n_i}$$

Where $\bar{x_t}$ is the smoothed position at frame t, k is the size on each side to consider for smoothing (i.e. larger k means a smoother curve), x_i is the averaged position at frame i, and n_i is the number of feature points in this group at frame i. We used a value of 2 for k. This smoothing is done for both x and y positions. The endpoints of the trajectory where $n_i = 1$ are cut off because a single feature point's trajectory rarely matches that vehicle's trajectory (if this last step is omitted, there are jumps at the beginning and at the end).

EXPERIMENTAL RESULTS

The system was evaluated based on the number of correctly counted vehicles. To calculate the accuracy, the numbered labels (like those on Figure 6) were manually identified (or noted) as being in one of the following three categories:

- True Positive

  It is tracking the vehicle correctly for most of the time it is visible on screen.

- False Positive

  It is either tracking part of the road without any vehicle, or tracking a vehicle that is already being tracked (over-counting).

- False Negative

  A vehicle passed by but no labels were assigned to it.

The following formula was used to calculate the accuracy:

$$Accuracy = \frac{TruePositive}{TruePositive + FalsePositive + FalseNegative}$$

The overall performance was assessed based on eight 30 seconds clips from the provided video which has a frame rate of 20 frames per second. Test sequences 1-4 are generally vertical motion (when seen from above) and tests 5-8 are horizontal motion. These clips were chosen since they appeared to be representative of the entire video. Counting performance is shown in Table 2 and a breakdown of the execution time is in Table 3.

The overall accuracy on this experiment was 56% and the system processed the video frames at a rate of 8.7 ± 1.4 frames per second with almost all of the time taken in trajectory extraction.

Table 2: Counting performance for each test.

Test No.	Correctly Counted (True Positive)	Missed (False Negative)	Over-counting (False Positive)
1	19	1	13
2	19	3	14
3	23	2	16
4	12	0	10
5	10	0	4
6	12	0	8
7	9	5	2
8	10	0	10
Total	144	11	77

DISCUSSION

We note that most of the errors are due to over-counting and that there are very few missed vehicles which suggests that: (1) our feature points are correctly tracking the vehicles, and that (2) the point trajectories need to be grouped together more aggressively. Part of the problem can be attributed to the fact that our rectification model assumes that all of the tracked points are close to the ground which is not always the case. Points that

Table 3: Total Execution time.

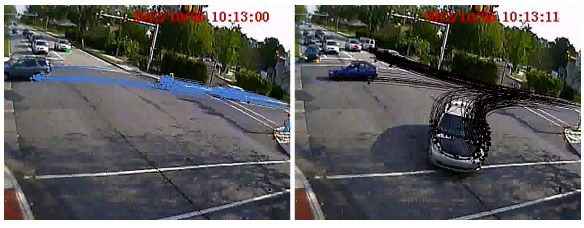
Computation	Time (sec)
Trajectory Extraction	68.576 ± 12.968
Trajectory Rectification	0.024 ± 0.007
Velocity Computation	0.004 ± 0.001
Trajectory Similarity Evaluation	0.334 ± 0.187
Trajectory Grouping	0.023 ± 0.011
Total	68.962 ± 13.174

are further from the ground plane are displaced by the rectification procedure and may become disconnected from their neighbors which leads to over- counting.

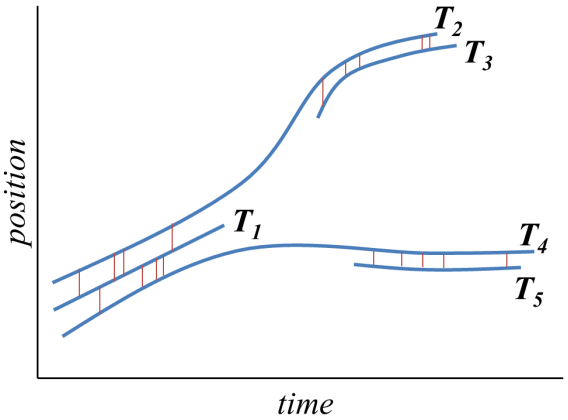
Another observation with regard to over-counting is that feature points are generally detected around the front and back sides of the vehicle but not in between. This may also contribute to detecting the same car multiple times. On the other hand, points are not detected within the vehicles' shadows. This suggests that feature point tracking is a robust approach to video-based vehicle tracking applications where shadows are present.

Over-counting may also be caused by occlusion. A vehicle's path may be split into two sections due to temporary occlusions, resulting in double-counting. When such a case occurs, the KLT tracker loses the features. This is difficult to handle during trajectory extraction, so it is probably better solved on a higher level, for instance by using a Kalman filter to track individual vehicles. This may also help solve the issue of re-counting once-lost vehicles, which occurs frequently when they are not moving.

We note that there are a few instances in which the program misses passing vehicles by over-grouping them. Such an error can occur when two or more vehicles move close to each other the entire time or when they start from the same position. The main cause can be attributed to Dulmage-Mendelsohn decomposition. This approach to grouping is brittle in that even a single connection between two groups of trajectories could make them count as a single group. Figure 7 illustrates such case.



(a) Two example cases in which vehicles are group together.



(b) A hypothetical example. Blue lines correspond to trajectories and red lines indicate connections between them. In this example two separate vehicles are grouped together.

Figure 7: Over-grouping example and causes.

CONCLUSION

In this paper, we present a technique to combine background subtraction and feature tracking to extract and group trajectories to determine traffic parameters. This method is robust to shadows and is able to handle perspective distortions to some extent. This traffic monitoring system was evaluated based on the number of cars taking one of 16 paths over eight 30 second video sequences. It has an accuracy of 56% and execution speed of 8.7 ± 1.4 frames per second.

The accuracy of this system may be improved by incorporating a better trajectory grouping technique as well as ways to account for occlusion. Execution time may be improved by focusing on the trajectory extraction step.

FUTURE WORK

There are three main areas worth investigating. The first is to take into account possible variations in feature heights when grouping trajectories. As mentioned before, the current method considers everything to be on the ground plane thus causing large positional errors on some areas of the intersection. It may be possible to guess the unknown height for each tracked point using its relationship with the others such that a more accurate trajectory may be obtained.

The second point is to cluster the trajectories using normalized cuts, a technique less brittle than Dulmage-Mendelsohn decomposition for determining subgraphs from adjacency matrix [14]. An alternative approach would be to group trajectories based on finding strongly connected cliques in the adjacency graph.

Acknowledgements: We are particularly grateful to Beau Kuhn for working with us over the summer and to T-SET for their financial support.

REFERENCES

[1] Miovision technologies. `http://miovision.com/`. Accessed: June 6, 2014.

[2] Technologies for safe and efficient transportation. `http://utc.ices.cmu.edu/utc/`. Accessed: June 6, 2014.

[3] Takashi Furuya. Public repository for the software for road intersection monitoring from video with large perspective deformation. `https://github.com/takfuruya/DVRPC`, 2014.

[4] Benjamin Coifman, David Beymer, Philip McLauchlan, J. Malik, and Jitendra Malik B. A real-time computer vision system for vehicle tracking and traffic surveillance, 1998.

[5] Chris Stauffer and W. E L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, pages –252 Vol. 2, 1999.

[6] S. Gupte, O. Masoud, R.F.K. Martin, and N.P. Papanikolopoulos. Detection and classification of vehicles. *Intelligent Transportation Systems, IEEE Transactions on*, 3(1):37–47, Mar 2002.

[7] A. Ghasemi and R. Safabakhsh. A real-time multiple vehicle classification and tracking system with occlusion handling. In *Intelligent Computer Communication and Processing (ICCP), 2012 IEEE International Conference on*, pages 109–115, Aug 2012.

[8] D.R. Magee. Tracking multiple vehicles using foreground, background and motion models. *Image and Vision Computing*, 22:143–155, 2001.

[9] D. Roller, K. Daniilidis, and H.H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, 10(3):257–281, 1993.

[10] A.M. Cheriyadat, B.L. Bhaduri, and R.J. Radke. Detecting multiple moving objects in crowded environments with coherent motion regions. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1–8, June 2008.

[11] N. Saunier and T. Sayed. A feature-based tracking algorithm for vehicles in intersections. In *Computer and Robot Vision, 2006. The 3rd Canadian Conference on*, pages 59–59, June 2006.

[12] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593–600, Jun 1994.

[13] Matlab klt point tracker. `http://www.mathworks.com/help/vision/ref/vision.pointtracker-class.html`. Accessed: June 6, 2014.

[14] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, Aug 2000.