Traffic Counting with Kalman Filters

Anthony M Lozano
Department of Computer Science
Georgia Institute of Technology
amlozano1@gmail.com

Abstract

Counting vehicles is an important function for traffic video data analysis. This paper examines an approach using a Shi-Tomasi corner detector with a Lucas Kanade Optical flow pyramid to create data measurements. After clustering tracked data measurements, a Kalman Filter is used to process the data in order to finally determine if a vehicle passes through the video. Our implementation of this method shows some success, but needs refinement.

1 Background/Problem Description

Accurate traffic data has diverse applications; Civil Engineers use it to design, construct, and maintain roadways. It is utilized by Policy Makers to allocate funding for such projects, and business owners may need it to make sound decisions about location. In addition, this information can be used to estimate the environmental impact of roadways and to perform other similar usage based calculations.

While the data is certainly useful, extant methods of collecting it are more expensive and cumbersome than necessary. The chief problems of current solutions are portability, setup time, and need for humans to process the data. Portability might not be an issue for areas where traffic data will need to be collected permanently, but usually agencies want a system that can move easily to take measurements from many sites. Some examples of less portable systems include inductive loop sensors installed into the roadway itself, or the TRAX Stealth Stud, which is another device that needs to be installed into the pavement. Installing these devices requires closing the road, which is not ideal in situations where even a relatively short road closure may cause major disruptions, such as urban areas. Pneumatic loop sensors, which look like small black tubes that are laid perpendicularly over the road, are an effective solution, but some jurisdictions in the United States require that workers close the road when they are putting these up anyway for safety reasons. Other systems use a camera to record video and a human later counts the cars going by, producing very accurate data at the cost of being labor intensive and time consuming. These systems and devices are also expensive, costing thousands of dollars for a single unit or many man hours of labor to operate a single unit

This paper attempts to create a software analysis tool that can perform one of the most basic tasks for traffic data collection; counting how many cars have gone in a particular direction in a video. This software tool would enable anyone with a camera to collect traffic data.

2 Hypothesis and Project Approach

Computer vision techniques for tracking motion in video combined with a Kalman Filter should be able to track vehicles and plot their movements. The software will then record the starting and ending points of each successfully tracked vehicle, thus establishing counts for each origin and destination.

3 Implementation Details

The Kalman Car Counter program will need to perform six major tasks to count vehicles. These tasks are background subtraction (or foreground detection), feature detection, optical flow estimation, optical flow clustering, and finally Kalman Filtering.

3.1 Background Subtraction

Background subtraction is the process of removing noise and unmoving sections of an image. The Kalman Car Counter uses the simple process of frame differencing to remove the background for speed. Frame difference at a time t can be calculated as $D(t+1) = V(x, y, t+1) - V(x, y, t)$ for each pixel (x, y) [7]. This calculation will allow the subsequent algorithms to focus solely on areas of the video that contain motion, helping isolate vehicles and removing image noise.

3.2 Feature Detection

Feature detection is the process of finding good features of an image for tracking. For vehicle tracking, Shi-Tomasi corner detection is a solid algorithm choice that expands on standard Harris corner detection, and will allow us to zero in on the "corners" of vehicles. Briefly, the algorithm works as follows: If we can assume the time between frames is small, we can describe the changes between them as image motion. If we take a patch of an image defined by (u, v) and shift it by (x, y), the weighted sum of squared differences between the two patches, S is given by

$$S(x,y) = \sum_u \sum_v w(u,v)(I(u+x, v+y) - I(u,v))^2$$

$I(u+x, v+y)$ can be approximated bu a Taylor expansion with I_x, and I_y being the partial derivatives of I such that:

$$I(u+x, v+y)) \approx I(u,v)(I_x(u,v)x + I_y(u,v)y)^2$$

which produces the approximation

$$S(x,y) \approx \sum_u \sum_v w(u,v)(I_x(u,v)x + I_y(u,v)y)^2$$

which in matrix form is

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix}$$

where A is the structure tensor

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

A corner is then found if S has a large variation in all directions of the vector $(x \quad y)$. Thus, if A has two positive eigenvalues that are large in comparison to other eigenvalues, a corner is found [8]. The Shi-Tomasi corner detector takes a practical shortcut and only calculates $min(\lambda_1, \lambda_2)$ rather than doing a complete eigenvalue decomposition in the interest of speed and because this is sufficient to find features with great accuracy [6].

3.3 Optical Flow

Once we have determined what features in the image to track, we actually need to track them. To accomplish this, the Kalman Car Counter will use another well known algorithm, the Lucas-Kanade method. This method assumes that pixels move a small amount between frames, and the movement of the pixels is approximatly the same for all pixels in the neighborhood of a point p [4]. Therefore, the optical flow equation should hold for all pixels in a window centered at p. This means the local velocity of a feature (V_x, V_y) must satisfy the equations:

$$\begin{gathered} I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2) \\ \vdots \\ I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n) \end{gathered}$$

where $q_1, q_2, \ldots, q_n$ are the pixels around a feature inside the window and $I_x(q_i), I_y(q_i), I_t(q_i)$ are the partial derivatives if the image I with respect to position and time x, y, and t

This system of equations can be written in the matrix form:

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

This overdetermined system is solved with a compromise solution by the least squares principle, solving the 2x2 system:

$$A^T A v = A^T b$$

Bouguet presents a pyramidal implementation of the Lucase Kanade tracker [1], which is the particular implementation that the Kalman Car Counter uses. If the Lucas-Kanade tracker tracks the features detected in the previous step successfully, we can use those features as data points for our Kalman Filter.

3.4 Optical Flow Data Clustering

The Lucas-Kande tracker can return many successfully tracked features for a single vehicle. We can leverage this by clustering the tracked features to combine them into a single point. Because we cannot predict how many vehicles will be in a frame, we use a agglomerative hierarchical clustering scheme. We use cophenetic distance t as a criterion for automatic clustering of the data, expecting different vehicle feature clusters to have a large distance between them. Once the data is clustered, the means of each cluster that has at least n elements is assigned to a Kalman Filter.

3.5 Hungarian Algorithm

Once each cluster's mean has been established, we intend to again track each cluster, but this time using a Kalman Filter (described in Section 3.6) in order to be more tolerant to noise, occlusion, and merged clusters. Thus, each cluster must be assigned to a Kalman filter. For each existing Kalman filter, we attempt to assign the nearest cluster to that filter for measurement updates. In order so solve the assignment problem we use the Kuhn-Munkres or, simply, Munkres algorithm [5]. It solves a cost matrix where $a1$ would be the distance from a cluster mean a to a filter's next prediction 1:

$$\begin{bmatrix} a1 & a2 & a3 & a4 \\ b1 & b2 & b3 & b4 \\ c1 & c2 & c3 & c4 \\ d1 & d2 & d3 & d4 \end{bmatrix}$$

Solving this cost matrix for the minimum cost produces assignments for each cluster. If the algorithm assigns a cluster to a Kalman filter, it is used as a measurement update for that Kalman filter for the frame. Note that this matrix can be rectangular; there could be more clusters than Kalman filters or more filters than clusters. If there are more Kalman filters than clusters, Kalman filters that do not get an assignment are marked as ”Lost” and do not receive a measurement update. Lost Kalman filters accumulate a Loss Counter each subsequent frame they are lost in until they are either deleted or are again assigned to a cluster. A filter can also be lost if it's best assignment is to far away. On the other hand, if there are more clusters than Kalman Filters, a new filter is created for each new cluster, instantiated with initial position equal to the cluster mean.

3.6 Kalman Filter

The final step solves a practical problem: even with all the previous steps, our measurements are still somewhat noisy measurements of the vehicles location in the video. In order to better track the vehicles, we pass the measurements through a Kalman filter. The filter allows us to track vehicles even if vehicles' measurements are noisy, vehicles are temporarily occluded, or vehicles' clusters merge.

The Kalman filter requires several matrices, the most important of which is the next state function. This Kalman filter will model the motion of the vehicles using a simple system F of (x, y) position and velocity.

$$F = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Read in simple English, this matrix means that the next state's position should be estimated using the prior position and velocity multiplied by the change in time. The change in time is set to the number of seconds between each frame of the video. For example, in a 30 frames-per-second video, dt would be .5 seconds. Each Kalman filter will have it's initial state X set to the mean of the cluster (x, y) that created it with no initial velocity, that is $V_x = 0$ and $V_y = 0$.

$$X = \begin{bmatrix} x \\ y \\ 0 \\ 0 \end{bmatrix}$$

our measurement update function will be

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Since we can measure x and y but not V_x or V_y the uncertainty for the velocity will be set very high, while the uncertainty for position will be set relatively low.

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 1000 \end{bmatrix}$$

The Kalman filter performs prediction step and calculates the posterior position X':

$$X' = FX + u$$

and it updates uncertainty by:

$$P' = F \cdot P \cdot F^T$$

Similarly, after each prediction step, a measurement update step must be performed with a new measurement Z.

$$\begin{gathered} Y = Z - H \cdot X \\ S = H \cdot P \cdot H^T + R \\ K = P \cdot H^T \cdot S^{-1} \\ X' = X + (K \cdot Y) \\ P' = (I - K \cdot H) \cdot P \end{gathered}$$

With these two steps iterating every frame, we track each vehicle's starting location to ending location. With the tracked vehicle starting and ending positions in hand, we can count vehicles entering the scene and leaving the scene.

4 Technical Specifications

The Kalman Car Tracker has been implemented in the Python programming language. This implementation uses several technologies to accomplish it's task. The foremost is `OpenCV` (Open Source Computer Vision Library), an open source computer vision and machine learning software library. [10] It contains many optimized and state-of-the art algorithms of use to this project with Python bindings. In particular, the background subtraction, Shi-Tomasi corner detection, Lucas-Kanade optical flow tracking, and Kalman filteings all come from the `OpenCV` library. The hierarchical data clustering algorithm comes from another powerful Python library known as `SciPy`. Munkres linear assignment comes from a additional `(Scipy)` library by Lars Buitinck.[3]

To test this method, a dataset was captured with a cellphone camera on a highway overpass. A video was recorded from the center of the overpass overlooking the freeway at a resolution of 1080p for three minutes. The overpass vibrated, introducing much noise for our particular background subtraction method. In addition, the camera was held in hand; no tripod or other stabilization was used to record video.

The video processing took approximately 15 minutes for 3 minutes of video on a Intel i7 960 CPU @ 3.20 GHz.

5 Analysis of Results

In the first half of the sample video on the left (oncoming) lanes of traffic, $24/35$, or about 68%, of vehicles were successfully tracked. Our definition of successfully tracked is that Kalman filter tracking the car was assigned for at least the last 6 frames of the car being in the image. Having the car successfully tracked when it leaves the frame would be sufficient to count the car as having flowed under the overpass. While clearly there is room for improvement, this shows the method implemented in this tool can successfully track vehicles and perform the counting task. The parameters for various methods were not optimized, and the tool is far from complete, but these preliminary results seem promising. Perhaps with optimization and refinement this method could be used as envisioned to enabled simple unattended capture of traffic data.

6 Conclusion

In this paper, we present a method for collecting traffic data using a simple camera and computer vision processing, even in the presence of occlusion. The method needs refinement, but can successfully track vehicles. This method seems to have trouble differentiating between clusters of tracked optical flows when they are tightly packed, which could prove a large problem for dense traffic flows. This problem might be solved with a better camera perspective, but the best camera perspectives, such as top down, make the detection process much more trivial and come with disadvantages similar to those provided in the introduction: they are costly and hard to set up. Future research could explore other options for clustering that may help alleviate this issue.

7 Presentation and Github

A quick 5 minute presentation of this implementation is available at `https://www.youtube.com/watch?v=CMG_3c9UbsE`.

A longer, more complete presentation is also available at `http://youtu.be/lEuYAGPUJWk`

The implementation source code is available at `https://github.com/amlozano1/kalman_car_counter` under the MIT License.

References

[1] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 2:3, 2001.

[2] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[3] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[4] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[5] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial & Applied Mathematics*, 5(1):32–38, 1957.

[6] 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.

[7] Birgi Tamersoy. Background subtraction.

[8] Tiziano Tommasini, Andrea Fusiello, Emanuele Trucco, and Vito Roberto. Making good features track better. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 178–183. IEEE, 1998.

[9] Wikipedia. Lucaskanade method — wikipedia, the free encyclopedia, 2014. [Online; accessed 7-May-2014].

[10] Wikipedia. Opencv — wikipedia, the free encyclopedia, 2014. [Online; accessed 8-May-2014].