AADECA 2016 - Semana del Control Automático - 25º Congreso Argentino de Control Automático
1 al 3 de Noviembre de 2016 - Buenos Aires, Argentina.

VEHICLE CLASSIFICATION AND SPEED ESTIMATION USING COMPUTER VISION TECHNIQUES

Agustín Yabo, Sebastián Arroyo, Félix Safar, Damián Oliva

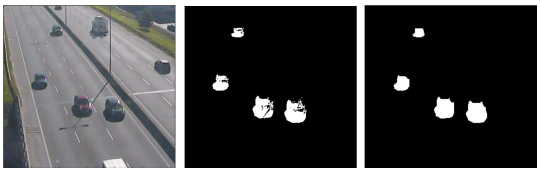
* *Universidad Nacional de Quilmes - doliva@unq.edu.ar*

Abstract: In this work, we implement a real-time vehicle classification and speed estimation system and apply it to videos acquired from traffic cameras installed in highways. In this approach we: a) Detect moving vehicles through background-foreground segmentation techniques. b) Compare different supervised classifiers (e.g. artificial neural networks) for vehicle classification into categories: (car, motorcycle, van, and bus/truck). c) Apply a calibration method to georeference vehicles using satellite images. d) Estimate vehicles speed per class using feature tracking and nearest neighbors algorithms.

Keywords: speed estimation, computer vision, traffic camera, feature tracking, vehicle classification

1. INTRODUCTION

Intelligent Transportation Systems (ITS) have, amongst its main goals, to avoid traffic delays and traffic jams, improve road safety, and reduce power consumption and emissions. Daily users and transportation agencies benefit from information supplied by ITS through improvements on traffic flow monitoring, management and control. The objective of the present work is to apply Computer Vision techniques to estimate vehicles georeferenced position and speed as part of an ITS. Some of the advantages of leaning towards a Computer Vision approach have already been established in prior work (Oliva *et al.*, 2015), in which we developed an algorithm for traffic flow description with no classification capabilities. In the present paper, we focus on the classification issue. In section 2, we describe the Computer Vision techniques and Machine Learning scheme used for the task. In section 3, we show the measurement results obtained from applying these methods to a video obtained from a highway traffic camera. Finally, in section 4, we discuss results and potential improvements to develop in the future.



(a) (b) (c)

Fig. 1. (a) Image acquired by a traffic camera. (b) Moving object detection by the background subtraction algorithm. (c) Result after applying morphological operations.

2. MATERIALS AND METHODS

The analysed videos were acquired at a 1 fps rate, with a resolution of 450x800 pixels, automatically obtained from an online IP camera (Autopistas del Sol S.A., 2014). An example of a typical frame is shown in Figure 1a. All the processing algorithms were implemented in Python 2 and OpenCV 3, both open-source projects.

2.1 Detection of moving objects with background subtraction

Background subtraction is a widely used approach for detecting moving objects. The concept behind

this algorithm is roughly described as detecting moving objects from the difference between current frame and a reference image, usually referred as *background image*. This background image should be a representation of the scene without objects in motion, and needs to be updated periodically in order to adapt to lighting changes. In this work, the *MOG* (Mixture of Gussians) algorithm is used (Stauffer and Grimson, 1999). MOG's main assumption is that the probability distribution of the intensity value x at time N is described as:

$$p(x_N) = \sum_{k=1}^{K} w_k \eta(x_N; \theta_k), \tag{1}$$

where K is the number of Gaussian distributions (which, in practice, is dynamically selected in the range of 3 to 5), w_k a weight parameter of the k^{th} Gaussian component and $\eta(x_N; \theta_k)$ the normal distribution of the k^{th} component, represented by:

$$\begin{aligned} \eta(x_N; \theta_k) &= \eta(x_N; \mu_k, \Sigma_k) = \\ &= \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(x_N - \mu_k)^T \Sigma_k^{-1} (x_N - \mu_k)}, \end{aligned} \tag{2}$$

where μ_k is the distribution mean and $\Sigma_k = \sigma_k^2 I$ the covariance of the k^{th} distribution. This implies that R, G and B pixel values are independently distributed and share the same variances, assumption that, although at the expense of a certain precision loss, reduces processing time considerably.

When a new image is acquired, the MOG algorithm acts iteratively on every pixel performing the following steps: 1) The pixel is assigned to the class with the mean value μ closer to the intensity value of the pixel. 2) Once it has been assigned to its closest class, the parameters estimation of the Gaussian distributions are updated according to the following equations:

$$\begin{aligned} \hat{w}_k^{N+1} &= (1-\alpha)\hat{w}_k^N + \alpha \hat{p}(\omega_k | x_{N+1}), \\ \hat{\mu}_k^{N+1} &= (1-\alpha)\hat{\mu}_k^N + \rho x_{N+1}, \\ \hat{\Sigma}_k^{N+1} &= (1-\alpha)\hat{\Sigma}_k^N + \\ &\quad + \rho (x_{N+1} - \hat{\mu}_k^{N+1})(x_{N+1} - \hat{\mu}_k^{N+1})^T, \\ \rho &= \alpha \eta(x_{N+1}; \hat{\mu}_k^N, \hat{\Sigma}_k^N). \end{aligned} \tag{3}$$

In order to select which Gaussian distributions describe the background model, the K existing components are ordered according to the fitness value w_k/σ_k, and the first B distributions are used as the background model, where B is estimated as:

$$B = argmin_b \left(\sum_{j=1}^{b} w_j > T \right), \tag{4}$$

where T is a threshold value that determines the probability that the analysed pixel belongs to the background model.

In this way, the algorithm decides whether a pixel is classified as background or as foreground. The values used in our implementation were $T = 0.7$ and $\alpha = 0.05$. Figures 1a and 1b illustrates an example of a raw acquired image, and the classification performed by the algorithm respectively. In the interest of reducing segmentation errors produced by noise, we used morphological operations. Specifically, Erosion using a square 2×2 kernel, and Close with a round 16×16 kernel were aplied (result in Figure 1c). It is noteworthy that quadrangular ROIs (Regions of Interest) were defined initially by an operator in order to simplify the analysis (as seen in Figure 2).

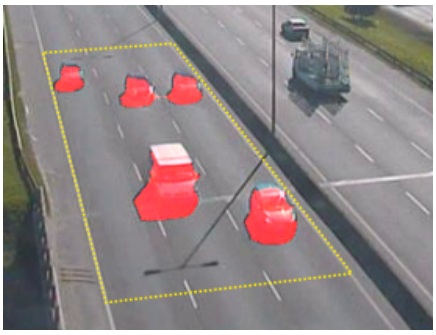


Fig. 2. Quadrilateral ROI delimited by the operator (dotted yellow line) and detected moving objects (red).

Road's Occupancy We define the road's occupancy for the specified ROI at time t as:

$$\rho(t) = \frac{A_{frg}(t)}{A_{ROI}}, \tag{5}$$

where A_{frg} and A_{ROI} are the number of pixels classified as foreground and the total amount of pixels analysed, respectively. This value varies from 0 to 1. In Oliva *et al.* (2015) we have shown that a relation between normalized average speed and road's occupancy exists, as it can be seen in the example of Figure 3. In this case, the occupancy remains under 0.2 most of the time, but when it increases over this value (intervals 30-40 and 50-55), average speed decays abruptly. This situation can be visually matched to a *traffic jam*.

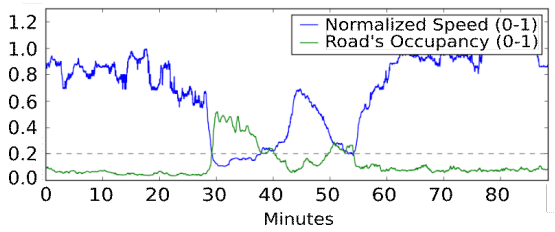


Fig. 3. Traffic analysis of a sample video. Speed is normalized in the range 0-1, being 1 the maximum speed allowed ($80 km/h$).

2.2 Camera calibration and Georeferencing

To be able to accurately measure the vehicles' speed, their cartesian position should be estimated in the *world* coordinates systems (X, Y) (in meters) in terms of their location (x, y) in the image (in pixels) (Figure 4).

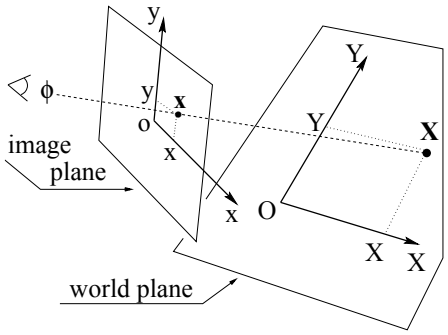


Fig. 4. (x, y) and (X, Y) axes represent the coordinate systems *image* and *world* respectively. Calibration points used to obtain the perspective transformation parameters match the quadrilateral vertexes in the image plane (Criminisi *et al.*, 1999).

Since the cameras we used can be modeled as *Pinhole* cameras, a perspective transformation can be adjusted to match both planes using quadrilateral ROI vertexes as calibration points:

$$X = \frac{ax + by + c}{gx + hy + 1}, \quad Y = \frac{dx + ey + f}{gx + hy + 1},$$

where $\lambda = (a, b, c, d, e, f, g)$ is a vector of parameters that describes the perspective transformation, and can be obtained through least squares using the pseudoinverse method (Criminisi *et al.*, 1999) on the quadrilateral vertexes. Figure 5 shows the quadrilateral in the *world* coordinate system, while Figure 2 shows the same quadrilateral seen from the traffic camera.

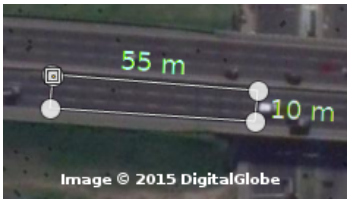


Fig. 5. Cartesian coordinates of the calibration points were measured using Google Earth software.

2.3 Feature tracking for speed estimation

In Oliva *et al.* (2015) we used OpenCV implementations of SURF (Bay *et al.*, 2006) and FLANN (Muja and Lowe, 2009) algorithms. SURF is applied to detect features and calculate its descriptors. It is because of SURF descriptors' robustness when facing scaling, rotations and light changes that we chose this algorithm in the first place. Thanks to this, it is possible to find and track the same feature in successive frames and study its movement (Figure 6), even in low frame rate videos, where the same vehicle changes position significantly from one frame to another.

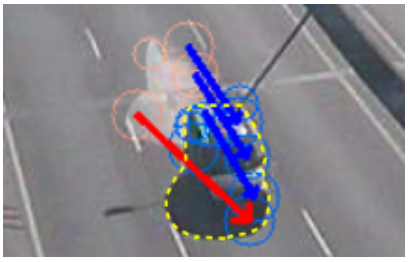


Fig. 6. Each arrow is obtained by feature tracking through SURF and FLANN algorithms. The red arrow indicates a match between features that has been filtered out by the angle filter.

The algorithm used consists of the following three steps:

(1) Detection: the algorithm computes SURF points of interest, and only those located inside blobs associated to moving objects are selected.
(2) Description: for each point of interest, a 64-dimensional vector is created representing a robust description of the point's neighbourhood in its characteristic scale (SURF descriptors).
(3) Matching: the problem of matching features can be simply described as finding those SURF descriptors' vectors with the minimum euclidean distance between them. This process is carried out by the nearest neighbours search algorithm FLANN. Moreover, only reliable matches are taken into account, defined as those matches in which its second nearest neighbour is 0.6 times farther than its first neighbour, according to Lowe (2004). Another criteria applied is a biunivocal filter through Forward-Backward error (Kalal *et al.*, 2010). We also encounter false positives matches (red arrow in Figure 6) that could be detected by applying a median filter to features' angles, excluding those with deviation greater than 5% from the median of all angles in the current frame.

The result of applying this method is a set of N_f pairs of features matched between frames at time N and $N + 1$. At this point, and after converting all features positions to *world* coordinates, we compute each vehicle's speed as:

$$v(t) = \frac{f}{N_f c} \sum_{n=1}^{N_f} \sqrt{\Delta x_n^2 + \Delta y_n^2}, \qquad (6)$$

where f is the acquisition rate of the videos in *frames per second*, $(\Delta x_n, \Delta y_n)$ the components of the displacement vector between features in the *world* coordinate system (in meters) and c a constant for conversion of units (m/s to km/h).

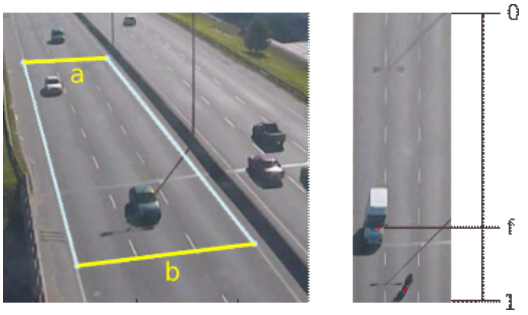


Fig. 7. Left: upper (a) and lower (b) dimensions of the quadrilateral. Right: relative position f in the range 0-1.

2.4 Vehicle classification

The main goal of this work is to develop an algorithm capable of classifying foreground blobs of an image into the following categories: car, bus/truck, motorcycle, van. After training a supervised classifier using classification information provided by the operator, the algorithm should be able to emulate the operator's choices. A graphical user interface was developed using wxPython library, providing the operator a simple way to classify each one of the vehicles detected. This allowed us to obtain more than 1000 blob samples and create a database, where extracted information from the blob as well as the manual classification performed by the operator were stored.

Blob descriptors For the current classification scheme, 3 different input parameters were chosen:

(1) Normalized area: due to perspective effects, each blob's measured area depends on the object's distance to the camera. To avoid this effect we compute a *normalized area*, using the area occupied by a blob and its location on scene:

$$A_{norm} = \frac{\sqrt{A_{measured}}}{(1 + f(\frac{b}{a} - 1))}, \tag{7}$$

where a and b are upper and lower dimensions (respectively) of the analysed quadrilateral, and f the relative position of the vehicle inside it, as illustrated in Figure 7. The normalized area is a characteristic size relative to the blob's position inside the quadrilateral. The value for a single object's A_{norm} remains essentially constant for every frame it is found in as it traverses the ROI.
(2) Circularity: it is defined as $f_{circ} = \frac{\sqrt{A}}{P}$, where A corresponds to the blob's area, and P to its perimeter (Corke, 2011).
(3) Aspect ratio: an ellipse is fitted to the blob using the Fitzgibbon algorithm (Fitzgibbon *et al.*, 1996) (as shown in Figure 8), and then, the ratio of its minor axis to its major axis is computed (Corke, 2011).

As explained before, the algorithm classifies depending on the type of vehicle. Additionally, an

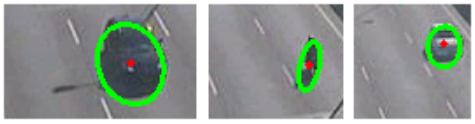


Fig. 8. Examples of ellipses fitted to blobs.

occlusion category was added to detect situations where the background/foreground algorithm was unable to separate two or more vehicles, leading to a single blob containing more than one object (see Figure 9). This is usually caused by shadow occlusion, noisy images, unwanted effects of the morphological operations, or simply visual occlusion. In this first approach we focused only on detecting the occlusions, since we have noted in the analysed videos that, most of the time, road's occupancy tends to remain under 0.2, condition in which occlusion rarely occurs (according to our observations, explained in Section 3).

Fig. 9. Example of vehicle occlusion due to perspective superposition.

Classification method In the process of selecting the most suitable classification method for this scheme, several supervised algorithms were tested and compared in *Python* using *scikit-learn*, a powerful open-source set of machine learning libraries that also includes data mining and data analysis tools. 10-fold cross-validation was used to obtain more accurate performance results, measuring in each experiment: 1) Training time. 2) Classification time. 3) Accuracy on train samples. 4) Accuracy on test samples.

Feed-forward neural networks This classification method consists of an input layer that receives a vector p of size $R \times 1$. For the present case, $R = 3$ since it is the number of parameters chosen to describe a single blob. The i^{th} layer of the neural network is comprised of S^i neurons with a a^i output state given by:

$$a^i = f^i(W^i \times a^{i-1} + b^i), \tag{8}$$

where W^i is the synaptic weight matrix of size $S^i \times S^{i-1}$, b^i the bias vector and f^i the activation function, as illustrated in Figure 10.

We implemented the neural network through *Neurolab*, a simple open-source library for Python, using the RPROP (Riedmiller and Braun, 1993) training algorithm, which, in our experience, has produced the best results (both in accuracy as in training time).

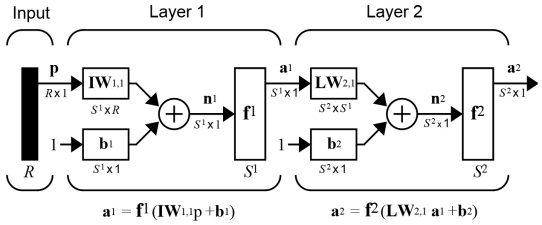


Fig. 10. Feed-forward multilayer Neural Network diagram.

3. RESULTS

3.1 Comparison of machine learning algorithms

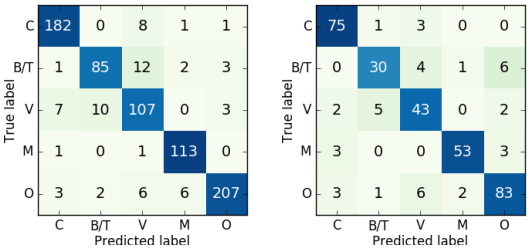
Table 1 shows the results of the classifiers comparison. As it can be seen, *Artificial Neural Networks* turned out to be the most accurate classifier for this case, followed by *Forest of randomized trees*, which, although it showed nearly the same performance in test samples, it performed in a significantly higher time of execution for each classification.

Each one of the tested methods was manually optimized for a sensible comparison. With regard to the ANN, the 10-fold cross-validation scheme mentioned in the previous section was also applied here to evaluate different neural networks geometries. We started from the simplest possible architecture, and began to subsequently increase its complexity (both in hidden layers as in amount of neurons) until a minimum error was reached. The parameter values specific for each method are:

ANN Hidden Layers: 2, Neurons: (15, 15, 5), Transfer functions: (LogSig, LogSig, SoftMax)
NNeighbours Neighbours number: 20, Weight function: uniform, Leaf size: 40
NCentroid Distance metric: mean
SGD Loss function: hinge, Penalty: standard l2, Alpha: 0.0001
SVM Penalty C: 1, Kernel: rbf
Decision Tree Criteria: gini, Splitter strategy: best
GBClassifier Loss function: deviance, Learning rate: 0.1, Stages: 100; Maximum depth: 3
Random Forest Estimators: 15, Criteria: gini
GaussianNB -

3.2 Classification stage

Figure 11 shows the classification results of one of the 10-fold cross-validation cases of the ANN, for train (11a) and test (11b) samples. It should be pointed out that the amount of samples showed in Figure 11 doesn't reflect the probability distributions for each class. We took special care to gather approximately the same amount of samples for each type of vehicle, so that the classifiers would train on each class in a balanced way.



(a) Train samples (b) Test samples

Fig. 11. Confusion matrices obtained from the artificial neural network approach.

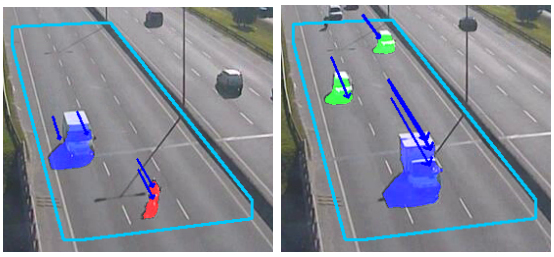


Fig. 12. The final implementation performing online color-coded classification and speed estimation.

3.3 Speed measuring per vehicle class

It was possible to measure an average speed for each type of vehicle classified by tracking features between frames and assigning each one of the measured distances to the vehicle's predicted category. A graphical example of this process is shown in Figure 12.

Using the classification routine we confirmed that, when road's occupancy is below 20%, occlusions are rare cases: after applying the developed algorithm to different online videos for more than 40 hours (with different light conditions and $\rho < 0.2$) we verified that only 6.28% of the blobs were classified as occlusions. It was also possible to obtain statistics on traffic flow speed per vehicle class. Figure 13 is an example of a traffic speed analysis performed with this method on a 1 hour long capture from a traffic camera. In practice, a median filter with a window size of 50 frames is applied to each curve, and in frames where samples for a particular class aren't found, previous speed values are kept. This explains why *Bus/Truck* class (the least frequent type of vehicle) fluctuates much less than the rest of the classes.

4. DISCUSSION

This work proposed an approach to estimating vehicles' speed over time by locating and tracking relevant features inside moving vehicles, and georeferencing them using satellite images. The use of features for vehicle tracking proved a

Table 1. Comparison of classifiers used.

Algorithm	Computation time [ms]		Average accuracy [%]	
	train	sample	train	test
Artificial Neural Networks	9851.1	0.289	92.51	84.66
Forest of randomized trees	86.040	1.867	99.47	84.35
Grading Boosting Classifier	883.54	0.681	92.11	84.06
Support Vector Machines	51.455	0.246	84.09	84.04
Gaussian Naive Bayes	2.891	0.519	85.41	83.74
Nearest Neighbours	2.660	1.909	86.33	82.82
Decision Tree learning	7.395	0.168	99.86	80.67
Nearest Centroid	1.723	0.389	79.50	78.83
Stochastic Gradient Descent	6.166	0.149	75.82	73.92

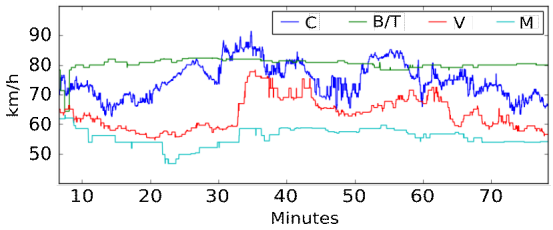


Fig. 13. Result obtained from analysing 60 minutes of a traffic camera. The graph shows 4 different speed curves, one per vehicle class.

robust method when facing light changes and image noise. Aiming for a segmented traffic flow description, different classifiers were tested in order to achieve the fastest and most accurate implementation. The results positioned ANN as the most effective algorithm for the current scheme. Altogether, feature tracking and ANN combined showed an efficient way to measure average speed per vehicle class, when analysing traffic videos with road's occupancy lower than 0.2. Since the overall system was designed taking into account the time cost of each algorithm, the resulting method was able to perform with a frequency of 10 to 15 fps, depending on the scene's occupancy, which allowed not only offline implementations applied to video files, but also *online* analysis of IP cameras. In the latter case, the speed bottleneck was mostly due to the network connection.

In future versions, we would like to develop an algorithm capable of dealing with occluded vehicles, so that crowded scenarios can be analysed in more depth. Another possible improvement could be to use multiple frames to classify each vehicle. Cross-matching each blob's classification with previous frames predictions could accomplish more accurate results, but this would require tracking blobs in addition to features, which, although feasible, is a completely different approach to the problem.

REFERENCES

Autopistas del Sol S.A., S.A. (2014). Autopistas del sol. `https://www.ausol.com.ar/transito.asp`.

Bay, Herbert, Tinne Tuytelaars and Luc Van Gool (2006). Surf: Speeded up robust features. In: *Computer vision–ECCV 2006*. pp. 404–417. Springer.

Corke, Peter (2011). *Robotics, vision and control: fundamental algorithms in MATLAB*. Vol. 73. Springer.

Criminisi, Antonio, Ian Reid and Andrew Zisserman (1999). A plane measuring device. *Image and Vision Computing* **17**(8), 625–634.

Fitzgibbon, Andrew W, Robert B Fisher et al. (1996). A buyer's guide to conic fitting. *DAI Research paper*.

Kalal, Zdenek, Krystian Mikolajczyk and Jiri Matas (2010). Forward-backward error: Automatic detection of tracking failures. In: *Pattern recognition (ICPR), 2010 20th international conference on*. IEEE. pp. 2756–2759.

Lowe, David G (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision* **60**(2), 91–110.

Muja, Marius and David G Lowe (2009). Fast approximate nearest neighbors with automatic algorithm configuration.. *VISAPP (1)*.

Oliva, Damián, Agustín Yabo, Lilián García, Sebastián I Arroyo and Félix G Safar (2015). Implementación de un sistema para la medición del flujo de tránsito y detección de embotellamientos en autopistas. In: *Argentine Symposium on Artificial Intelligence (ASAI 2015)-JAIIO 44 (Rosario, 2015)*.

Riedmiller, Martin and Heinrich Braun (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In: *Neural Networks, 1993., IEEE International Conference on*. IEEE. pp. 586–591.

Stauffer, Chris and W Eric L Grimson (1999). Adaptive background mixture models for real-time tracking. In: *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*. Vol. 2. IEEE.