Politecnico di Milano

Dipartimento di elettronica, informazione e bioingegneria

Image analysis

Software for traffic violation of a right hand rule

Author:
Mirjam Skarica,
Lucija Megla

Supervisor:
prof. Vincenzo Caglioti

July 11, 2016

Contents

1 Introduction

In the last few decades there has been a huge growth of traffic in the big cities. In consequence with growth of traffic, number of violations increased. As a result, software for traffic violation detection is a way to maintain order and track violations in a profitable way.

1.1 Motivation and formulation of the problem

Even though many crossroads have traffic lights or traffic signs, there are still crossroads without any of it. As many know, in that particular case right-hand rule is applied. In absence of traffic lights or signs, right hand rule demands from a driver of a vehicle to give away the priority to vehicles approaching from the right at intersections.

Given the above problem, the goal of the project is to develop a software that is going to recognize violations of the right hand rule and inform the operator about it. This way, we could gain an automated way to recognize this type of violations, which could help track situations on every intersection without traffic lights or signs. This is going to make tracking of right hand rule violations far more easier and profitable.

1.2 Camera and acquisition of video material

Formulation of the project for developing software for right-hand rule violations guarantees acquisition of the video material through a traffic surveillance camera placed in an elevated area. Since we did not have a surveillance camera at our disposal, we used a classical digital camera attached to an elevated object. Our camera had a frame rate of 30 fps and was attached to a trivet to reduce vibrations and other disturbances in order to capture a credible situation.

1.3 Document structure

This document serves to describe the structure of the software and its implementation in detail. Also, a detailed utility manual is given to describe software use-cases and to give a glimpse of situations in which software will work and some situations

in which might fail.

The report is divided into two sections:

1. Structure of the software and its implementation, in which we describe in detail approached used in development of our software;

2. Results, in which we give a critical review of our software and we list possible improvements of it.

2 Structure of the software and its implementation

To get a software that recognizes traffic violations of a right hand rule, we had to divide the task in more subtasks to get the desired behaviour. Those subtasks can be seen in Figure 1.

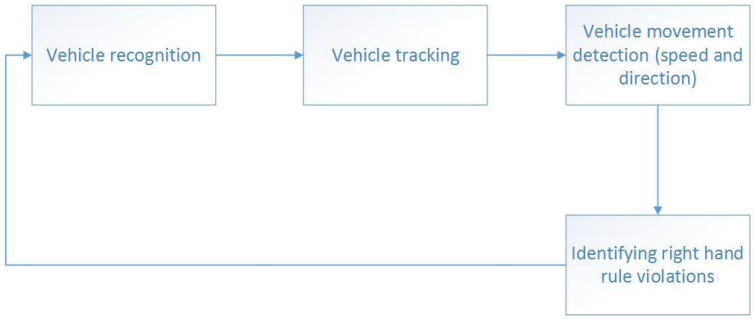

Figure 1: Software workflow

As we can see from Figure 1., first step in our algorithm consist of car recognition. In this step we had to extract background from the foreground and recognize cars in the given foreground. After extracting vehicles from the video footage, we had to track them through frames, so that we could determine their speed and direction. To identify if there is a violation of the right hand rule in the intersection, we divided the intersection in priorities and with the help of car's direction and speed determined if the violation occurred or not.

2.1 Vehicle recognition

When we look at problems video surveillance could solve, we need to make sure the system is robust. To say the system of video surveillance is robust, it should not depend on correct position of the camera and it should not lose its performance when there are:

- objects that can be partially seen in the scene
- overlapping objects in the scene
- shadows
- changes of lightning
- objects that move slowly
- objects that were visible and then removed from the scene

Talking about recognizing of vehicles, the problem of their identification consists mostly of being able to keep good estimation of the foreground. After that, applying filters to foreground estimation will give us a foreground with more quality in which we can extract vehicles with *blob analysis*. In the next few paragraphs we are going to explain all three steps in detail.

2.1.1 Foreground estimation

Major part of foreground estimation approaches are based on background estimation, which do not apply on our problem, that is, they did not give good results. That is why we chose to work with Gaussian Mixture Models (GMM), which gave us flexibility and robustness to confront all the problems stated above. Although different methods of GMM exist, like the one of Friedman and Russell ([FR97]), which explicitly classifies values of pixels with three different distributions and determines if the pixels belong to the street, shadows or vehicles, the method implemented in our software is the one of Stauffer and Grimson [SG99]. This method of GMM does not model all the pixels with one particular distribution, but models the value of one pixel with mixture of different Gaussians; the motive for using mixture of different Gaussians lies in the fact that multiple surfaces often appear in

the view frustum of a particular pixel and the lighting conditions change. Hence, mixture of Gaussians is able to surpass those changes. Each time the parameters of the Gaussians are updated, the Gaussians are evaluated using a simple heuristic to hypothesize which are most likely to be part of the "background process". Pixel values that do not match one of the pixels "background" Gaussians are grouped using connected components. Finally, the connected components are tracked from frame to frame using a multiple hypothesis tracker. This method uses two parameters: α and T. Parameter α represents learning rate and parameter T is a threshold for determining which distributions belong to the background and which do not.

We consider the values of a particular pixel over time as a "pixel process". The "pixel process" is a time series of pixel values, e.g. scalars for gray values or vectors for color images. At any time t, what is known about a particular pixel, x_0, y_0, is its history:

$$\{X_1, X_2, ..., X_t = I(x_0, y_0, i) : 1 \leq i \leq t\} \tag{1}$$

where I is the image sequence. The recent history of each pixel, $X_1, ..., X_t$, is modeled by a mixture of K Gaussian distributions. The probability of observing the current pixel value is

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,k} * \eta(X_t, \mu_{i,t}, \sum_{i,t}) \tag{2}$$

where K is the number of distributions, $\omega_{i,t}$ is an estimate of the weight (what portion of the data is accounted for by this Gaussian) of the i-th Gaussian in the mixture at time t, $\mu_{i,t}$ is the mean value of the i- th Gaussian in the mixture at time t, $\sum_{i,t}$ is the covariance matrix of the i-th Gaussian in the mixture at time t, and where η is a Gaussian probability density function:

$$\eta(X_t, \mu_{i,t}, \sum_{i,t} = \frac{1}{(2\pi)^{\frac{n}{2}} | \sum |^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t - \mu_t)^T \sum^{-1} (X_t - \mu_t)} \tag{3}$$

K can take values between 3 and 5. In this case, 5 is used. Also, for computational reasons, the covariance matrix is assumed to be of the form:

$$\sum_{k,t} = \sigma_k^2 \mathbf{I} \tag{4}$$

This assumes that the red, green, and blue pixel values are independent and have the same variances. While this is certainly not the case, the assumption allows us to avoid a costly matrix inversion at the expense of some accuracy. Thus, the distribution of recently observed values of each pixel in the scene is characterized by a mixture of Gaussians. A new pixel value will, in general, be represented by one of the major components of the mixture model and used to update the model.

Once we defined the model of the background, it is necessary to initialize diverse parameters of Gaussian mixtures. To avoid costly performance, Stauffer and Grimson [SG99] used K-means algorithm to initialize those parameters. Once we defined those parameters, it is possible to start the foreground estimation. To model this, we need a method for deciding what portion of the mixture model best represents background processes. First, the Gaussians are ordered by the value of ω/σ. This value increases both as a distribution gains more evidence and as the variance decreases. This ordering of the model is effectively an ordered list, where the most likely background distributions remain on top and the less probable transient background distributions gravitate towards the bottom and are eventually replaced by new distributions.

The first B distributions are chosen as background model, where:

$$B = argmin_b(\sum_{k=1}^{b} w_k > T) \tag{5}$$

where T is a measure of the minimum portion of the data that should be accounted for by the background. This takes the "best" distributions until a certain portion T, of the recent data has been accounted for. The other distributions are taken as foreground distributions. Hence, when a new frame comes in time $t+1$, match test is being held on each pixel; one pixel belongs to a Gaussian distribution if a Mahalanobis (6) inequality is satisfied:

$$sqrt((X_{t+1} - \mu_{i,t})^T \sum_{i,t}^{-1}(X_{t+1} - \mu_{i,t})) < k\sigma_{i,t} \tag{6}$$

where $k = 2.5$.

With the above equation there are two possible cases:

1. Pixel and one of K Gaussian distributions were matched. In this case, a pixel is classified as a background pixel if its distribution is marked as a

Name	Value
Num. Gaussians	5
Num. frames for learning	50
T	0.005
α	0.2

Table 1: Table of parameters for foreground estimation

background distribution, or as a foreground pixel otherwise.

2. If we did not find a distribution for a pixel, then the pixel is classified as a foreground pixel.

The result of this procedure on every pixel is a binary matrix which has a value 1 denoting a foreground pixel and a value 0 denoting a background pixel. In Table 1 we can see parameters for foreground estimation used in our software.

2.1.2 Filter application

Once we estimated the foreground, it was necessary to apply a filter to remove the noise. First we used function *imopen* which performs morphological opening on the foreground estimation with a square structure element having dimensions 3×3. After that we used function *imclose* which performs morphological closing on our estimated foreground. With function *imfill* we performed hole filling on our foreground to get better precision.

In Figure 2 we can see how the foreground looks like after the extraction and noise removal.

2.1.3 Blob analysis

After extracting the foreground and removing noise, we were ready to extract connected regions as blobs and identify which of the blobs correspond to vehicles. We performed blob analysis with MATLAB's BlobAnalysis. The output of blob analysis contains identified blobs which could correspond to vehicles. To remove possible outliers, we extracted width and hight of the blob and put a constraint over

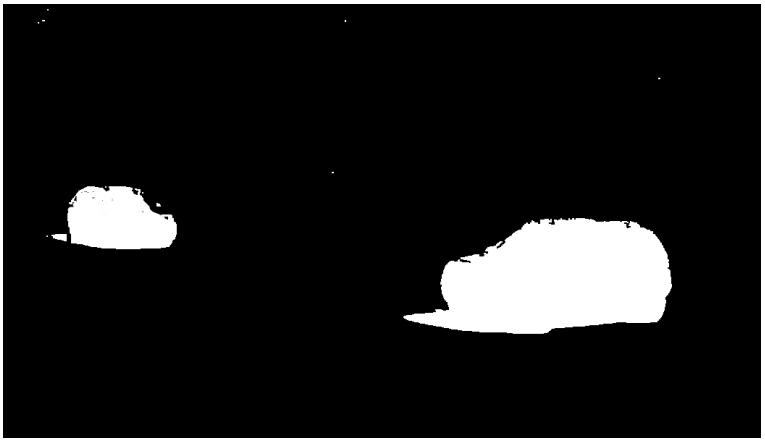

Figure 2: Extracted foreground with noise removed

the ratio between the two, which had to be greater than 0.9. If the corresponding blob was detected as a car, we saved its bounding box and centroid.

In Figure 3 we can see how the blobs are represented when processing the videos.

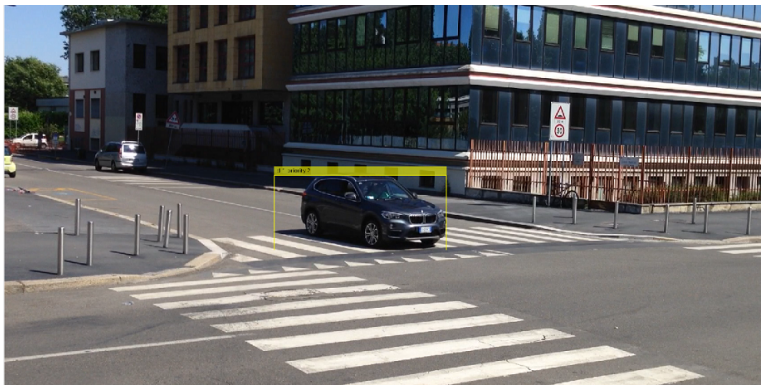

Figure 3: Detecting cars

2.2 Tracking of vehicles

In order to be able to recognize traffic violations, after foreground detection and blob extraction, we had to be able to track moving vehicles through certain period of time. In this part of our software we were focused strictly on motion.

The problem of motion-based object tracking can be divided into two parts:

1. detecting moving cars in each frame

2. associating the detections corresponding to the same car object over time

We described detection of moving cars in the previous subsection, where we used a foreground detection algorithm based on Gaussian mixture models, morphological operations to eliminate noise and blob analysis to detect groups of connected pixels, which represented moving vehicles.

The association of detections to the same moving car (i.e track) is based solely on motion. Each track is representing a moving car in the video. The purpose of the structure is to maintain the state of a tracked car. The structure contains the following fields:

- id: the integer ID of the track
- age: the number of frames since the track was first detected
- bbox: the current bounding box of the object; used for display
- bbox_initial: first bounding box
- velocity: car's speed
- priority: car's priority; used for violation detection
- kalmanFilter: a Kalman filter object used for motion-based tracking
- totalVisibleCount: he total number of frames in which the track was detected (visible)
- consecutiveInvisibleCount : the number of consecutive frames for which the track was not detected (invisible).

The motion of each track is estimated by a Kalman filter. This filter is used to predict the track's location in each frame and determine the likelihood of each detection being assigned to each track.

Track maintenance becomes an important aspect of this example. In any given frame, some detections may be assigned to tracks, while other detections and tracks may remain unassigned. The assigned tracks are updated using the corresponding

detections. The unassigned tracks are marked invisible. An unassigned detection begins a new track.

Each track keeps count of the number of consecutive frames, where it remained unassigned. If the count exceeds a specified threshold, the example assumes that the object left the field of view and it deletes the track.

The part of our software which is in charge of tracking moving cars can be divided in several parts:

- detection of moving cars in the current frame
- prediction of new locations of existing tracks
- assignment of detections to tracks
- update of assigned tracks
- update of unassigned tracks
- deletion of lost tracks
- creation of new tracks

For *predicition of new locations of existing tracks* we used Kalman filter. In this section, Kalman filter predicts the centroid of each track in the current frame and updates its bounding box accordingly.

Assignment of detections to tracks is done by minimizing cost. The cost is defined as the negative log-likelihood of a detection corresponding to a track. This procedure involves two steps:

1. Compute the cost of assigning every detection to each track using the distance method of the vision.KalmanFilter system object. The cost takes into account the Euclidean distance between the predicted centroid of the track and the centroid of the detection. It also includes the confidence of the prediction, which is maintained by the Kalman filter. The results are stored in an $M \times N$ matrix, where M is the number of tracks, and N is the number of detections.

2. Solve the assignment problem represented by the cost matrix using the *assignDetectionsToTracks* function. The function takes the cost matrix and the cost of not assigning any detections to a track.

The *assignDetectionsToTracks* function uses the Munkres' version of the Hungarian algorithm to compute an assignment which minimizes the total cost.

Update of assigned tracks function updates each assigned track with the corresponding detection. It calls the *correct* method of vision.KalmanFilter to correct the location estimate. Next, it stores the new bounding box, and increases the age of the track and the total visible count by 1. Finally, the function sets the invisible count to 0.

Function *update unassigned tracks* increases the age of invisible track by 1, so that the function delete lost tracks can delete tracks that were invisible for too many consecutive frames. It also deletes recently created tracks that have been invisible for too many frames overall.

2.3 Traffic violation recognition

The main goal is detection of the "right hand side" rule violation. When two vehicles find themselves in an intersection not governed by either a traffic light or a sign, then the rightmost vehicle as the priority of passage. The case when the other vehicle doesn't stop to let the rightmost pass is considered a violation of the *rhs* rule.

First step in detecting violation is detecting moving cars and assigning them priorities. More specifically, rightmost cars have the highest priority of 3 and the rest are assigned priorities in descending order. Detecting the priorities depends on the position of the camera. Having a still camera, we identified 3 possible roads each car could come from. This was done by 3 different polygon areas as shown in Figure 4. In determining priorities, only the first frame in which the car has appeared determines its priority. For example, if the car's bounding box coordinates fall within the green polygon, it will be assigned priority 2.

Having the system set up in this way, it's trivial to see that the only right-hand side rule violation could happen between cars of priority (1, 2) and (2,3). Next we track the cars and in each frame check for possible traffic violation between those the priority pairs. This is is done within functions *determineIfViolaitonP12* and

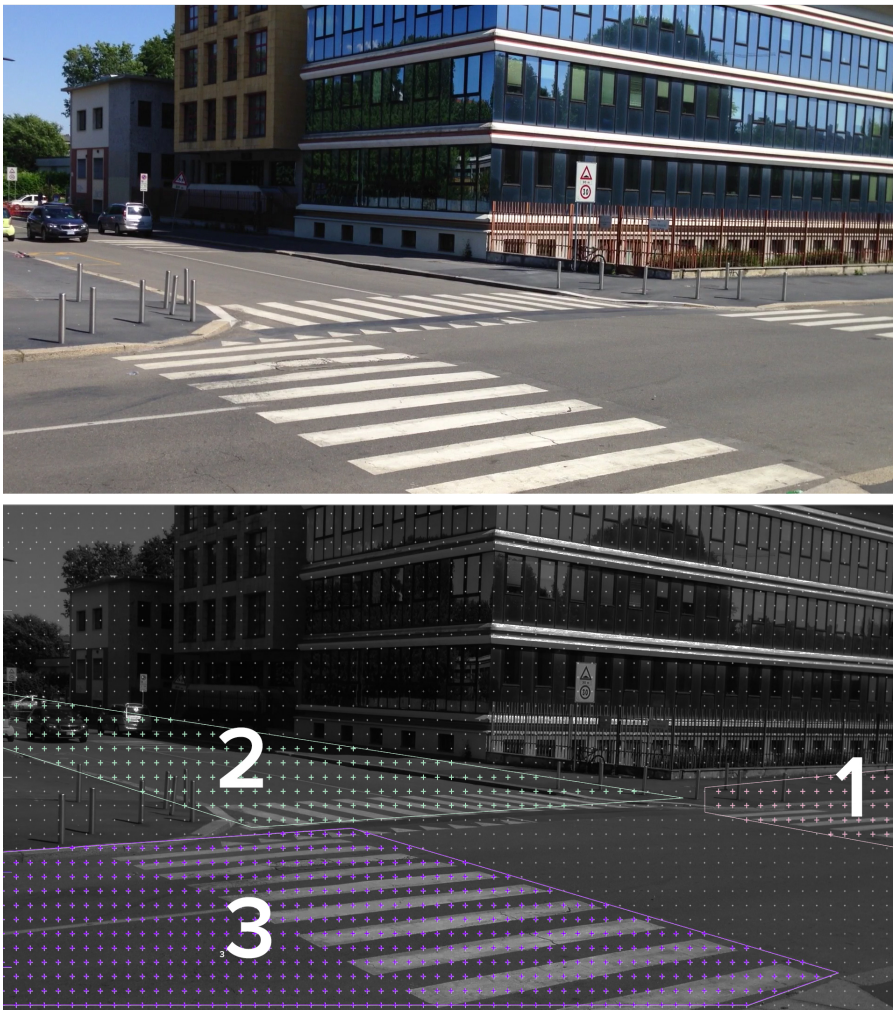

Figure 4: Detecting car's priority

determineIfViolaitonP23. In order for the program to decide if a violation between cars with priority 1 and 2 has happened these conditions need to be satisfied:

- the distance between the left bounding box edge of vehicle with priority 2 and the one with priority 1 has to be greater than $minDeltaDist = 10$. Meaning we make sure the cars have passed each other
- the distance between the top bounding box edge of vehicle with priority 1 and the one with priority 2 has to be greater than than $minDeltaDist = 10$. Meaning we make sure the car with priority 1 didn't make a left turn and is

now "above" the car with priority 2

- the overlap ratio of their bounding boxes has to be greater than *minOverlap = 0.2*.

- the difference between velocity [1]of the vehicle with priority 2 and the one with lower priority has to be greater than *minDeltaVelocity = 250*. Meaning, the vehicle with the higher priority has to have a much lower velocity, as though it was stopping or slowing down

2.4 Manual

The videos on which we have tested our software can be fond in the *video_data_sunday* folder. The software can be run for any video by specifying its path with the *videoPath* variable in the function *main*. Running it runs the the traffic detection violation program which outputs its result in the console.

3 Results

The results obtained after testing each video are the following:

- *video_data_sunday/IMG_5108.mov* the software correctly decides the first encounter of vehicles with priority 1,2 is not a violation due to their overlap ratio not being sufficiently large. Furthermore, it also correctly decides the second encounter was indeed a violation

- *video_data_sunday/IMG_6914.mov* an example of a car with priority 2 respecting the right-hand side rule, and stopping to let the car with priority 3 pass. The software correctly handles this situation

- *video_data_sunday/IMG_6915_01.mov* an example of traffic violation between two motorcycles with priority 1 and 3 which the software detects correctly

[1]The term velocity here is used somewhat loosely, because the true velocity could have been calculated only in case the actual intersection section dimensions were known, and if the center of the bounding box was less volatile. The calculation used for this velocity indicator can be found in the function *updateVelocity*

- *video_data_sunday/IMG_6917.mov* in this video, the first car with priority 1 makes a left turn and the software correctly decides it is not a violation. But when a second car with priority 1 passes the car with priority 2 making it a violation, the bounding box unexpectedly "jumps" from the first to the second car, and they end up with each others priorities. As a result, the software doesn't detect the violation

- *video_data_sunday/IMG_6919.mov* An example of one non-violation between (1,2) and one violation between (1,2) which software correctly detects

Unfortunately, because of the conditions regarding video recording, we weren't able to procure a video where a violation of the right-hand side rule between vehicles with priority 2 and 3 had happened. Hence we were not fully able to test it for that particular instance. In addition, we feel obligated to point out that even the slightest shake of a camera can cause the surrounding background to be detected as a moving object, which consequentially renders the software useless in traffic violation detection. Furthermore, all the videos we have tested are quite simple cases without much noise like pedestrians or too much traffic at the same time. In the future, it would be recommended to further refine motion detection so it also handles more complex cases.

References

[Figueredo and Wolf, 2009] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.