Occlusion Robust Vehicle Tracking based on SOM (Self-Organizing Map)

Alessandro Bevilacqua Luigi Di Stefano Stefano Vaccari
DEIS – ARCES (Advanced Research Center on Electronic Systems)
University of Bologna
Via Toffano, 2/2 – Bologna, ITALY, 40125

Abstract
Traffic monitoring systems based on image and sequence analyses are widely employed in Intelligent Transportation Systems (ITS’s) in order to analyze traffic parameters and statistics. To this purpose, tracking objects is often needed. However, occlusions can mislead a vehicle tracking system based on a single camera, thus resulting in tracking errors. In this work we present a vehicle tracking algorithm based on the KLT feature tracker which exploits a Kohonen Self Organizing Map (SOM) to drastically reduce tracking errors arising from occlusions, thus increasing the overall robustness of the system. Our method has been implemented in a real-time traffic monitoring system that has been working on daily urban traffic scenes. The experimental results we present assess the effectiveness of our approach even in the presence of quite congested traffic situations.

1. Introduction
Nowadays many traffic monitoring systems (TMS’s) rely on computer vision techniques in order to analyze statistics on traffic flows or to detect traffic congestion. To this purpose, these systems must be able to detect and track vehicles. Often, they rely on a distributed network of monocular cameras and the presence of occlusions among objects is one of the most challenging problems to deal with. As a matter of fact, when one camera is used and two or more vehicles overlap tracking them separately is a daunting task because the 2-D scene viewpoint could not permit a correct separation between moving objects. Some approaches coping with occlusion handling include 2-D or shape-based models. However, when a car turns in depth, we can have self-occlusion: 2-D model-based trackers will fail in this case, while shape-based methods are likely to lose track of the car because of the fast shape changing speed. In order to overcome most of these problems, other methods use feature points to track vehicles. In fact, even when vehicles are partially occluded, some of their feature points are still visible. Nevertheless, feature points may “jump” between vehicles and mislead a feature tracker. A probable solution could be exploiting more information or employing higher level methods. As a matter of fact, some other research works are based on 3-D models: they require camera calibration and a priori knowledge about objects that will move on the scene. Also, algorithms are often computationally expensive and cannot cope with real-time applications.

The work we present deals with a feature-based algorithm that tracks every object as a set of feature points. The system does not rely on any a priori knowledge about the targets within the scene. The approach we devised permits to improve the robustness of the KLT tracker by employing a Self-Organizing Kohonen Map (SOM) to detect feature wrongly tracked, thus reducing vehicle tracking errors. The algorithm has been implemented and embedded in a TMS working in an urban environment. The experiments accomplished by using challenging sequences showing urban road intersections give excellent performance in terms both of quality and computational time.

Section 2 analyses previous works regarding object tracking. Section 3 outlines the basic tracking algorithm. Section 4 explains how the SOM is used to reduce the errors of the tracking system. In Section 5 extensive experiments accomplished on both toy and real sequences are discussed. Finally, Section 6 draws conclusion and gives some lines for future work.

2. Previous Work
Many methods are known to track vehicles which use features and manage occlusions. In [1] features are used to track vehicles in highways, by grouping sets of features with the same motion. Homography combined with camera calibration is used in the grouping process to take road perspective into consideration. Features are tracked in consequent frames by a Kalman filter. This is also feasible due to the roughly constant speed of vehicles in a highway. Then a template matching is performed, searching for a correlation peak. Vehicles are tracked also if partially occluded by tracking feature points that are still visible. Features are grouped by distance or by relative speeds, but during the grouping process coefficients could cause errors in object
tracking due to over-grouping or unnecessary segmentation. Besides, the proposed method can work only with highway traffic, so it is not applicable in urban traffic intersections.

The authors in [2] propose a technique for removing outliers from the trajectories of feature points fitting a subspace and removing those points that have large residuals. They generate and track features through the entire video stream using the KLT algorithm, then apply the RANSAC method to detect trajectories that does not follow the rigid movement constraint. The latter is applicable when it is reasonable to assume an orthographic view with a weak perspective. So this algorithm fails when movements involve strong perspective changes, limiting its effectiveness only to short sequences. Also, the computational time needed to remove outliers is high and not compliant with real time processing.

In [3] tracking is performed by template matching, with a probabilistic model based on a robust error norm. Matching is performed by finding the image region that yields the maximum likelihood with respect to the calculated probability distribution. Templates are matched by translation, rotation and scaling parameters; during occlusions, a pixel is regarded as an outlier if the measurement error exceeds a predefined threshold. Templates cannot be updated during occlusion, so it is necessary to detect when overlapping ends. This algorithm can manage complete occlusions, but it cannot withstand severe occlusions lasting more than 25 frames, due to wrong Kalman filter estimation.

Authors in [4, 5] propose an occlusion robust method based on Spatio-Temporal Markov Random Field Model. Each frame is divided in blocks of $8 \times 8$ pixels and the feature correlation between blocks of consecutive images is performed by exploiting motion vectors for each block. In order to determine which object a block belongs to, a complex minimization problem must be solved. To this purpose a relaxation algorithm (Metropolis) is employed, thus preventing the feasibility for the system to work in real time.

In [6] authors propose to track vehicles as a set of parts to resolve partial occlusions. To fulfill the tracking purpose a specific segmentation method is proposed, which make use of active contours (“snakes”). Moving objects are initially segmented into a square grid, then contours are moved to minimize an energy function. The goal is to obtain parts that are the most suitable for a gradient-based tracking algorithm. However, presented results show only occlusions for cars that move out of the field of view, and only for one car at a time. Segmentation should also be repeated as vehicles change their appearance, resulting in a consistent computational overhead.

3. The basic tracking algorithm

Our tracker works on moving regions (“blobs”) previously extracted by a motion detector through a background difference approach. Inside each blob we extract corner points by applying the Shi-Tomasi method [7] and then we track those features by using the pyramidal KLT approach [8]. We derived the Lucas-Kanade equation as in [9], even though a more efficient method is proposed in [10]. In fact, it cannot be applied to our case because we estimate only translation parameters to track corner points, while optimizations presented in [10] focus on feature trackers dealing with rotation and scaling as well. As a matter of fact, given sufficiently high a frame rate, those transformations between consecutive frames are negligible for small features.

However, some features may be lost during tracking. Those features are replaced by using a minimum density approach: that is, each blob is divided in small squares of the same size and each square is tested to check whether the number of currently tracked features is over a threshold. If not, we perform feature extraction in every square that does not reach the threshold value.

3.1. From features to objects

To track objects starting from features, we need to group sets of corner points that are located on the same object. Possible ways to do that include grouping by speed, or by statistics given by feature position. However, the former is not always reliable: vehicles rotating on depth present features with very different speeds, that depend from trajectory curvature and point of view. Also, thresholds used for grouping are not known a priori. The method based on statistics tries to group set of features having the same motion over a period of time. In this case, perspective and self-occlusions cannot permit a correct grouping. While the first problem can be managed by homography, as in [1], the point of view can distort statistics for the features placed on an object rotating on the vertical axis.

Therefore, we perform extraction and grouping on a spatio-temporal basis. We can now distinguish 3 different labeling situations.

The first is when a new blob enters the scene, and no features have been extracted yet. As soon as an object appears, we extract features by assigning a unique number to all of them, that identifies the object during the whole tracking process. If a new blob enters the camera field of view in consequent frames, new features are extracted just from the last entered part. Those new features get the label of the nearest corner point already labeled within the same blob.

The second situation is when two objects enter the scene sharing the same blob, and so the same label. When they split, thus resulting in two distinct blobs, the algorithm creates a new label and renames one of the two blobs in order to identify an object from the other.

The third situation occurs during occlusions. When two or more objects previously distinct merge in one blob, obviously we find more than one label inside this new blob. To
maintain minimum feature density in the blob area (as explained in Section 3) new features have to be extracted and labeled reliably during all the occlusion process. To this purpose, a label is assigned only to features distant from the occlusion boundaries to avoid uncertainty about label assigning. In fact, within those regions features with different labels are close together, thus reducing label reliability. Instead, far from those areas each feature can be reliably labeled with the same label of the nearest object.

During all the tracking process, our system is able to dis-occlude each object by grouping features having the same label. At visualization level each group of features is delimited by a bounding box showing also the label of the tracked object.

### 3.2. Some problems

During occlusions, in the uncertainty region where objects overlap, some features could “jump” on the wrong object, causing a tracking error. It occurs when the KLT algorithm finds the correspondence between features in consecutive frames but it fails to track the right spot on the image. Objects therefore cannot be tracked correctly, because when occlusion ends we could have features previously assigned to a specific object attached to a different one. A feature tracking error thus becomes an object tracking error, meaning unreliable trajectory determination and false positive detection. In particular false positives are non-existent objects created after blob splitting, when a label is created to avoid the presence of the same object in two separate regions given by features jumped on the wrong object.

Feature density represents another key issue. Lowering the number of features could reduce errors, but it would be easier to lose track of an object, especially if small. Thanks to our error detection algorithm, the system we present can use high feature density (e.g.: to follow small or occluded vehicles) being robust at the same time. We apply SOM in order to detect and remove anomalies in feature speed vectors, caused by features jumped from an object to another one.

### 4. Using SOM to reduce errors

#### 4.1. SOM: an outline

Neural networks occupy a wide branch of Artificial Intelligence, involving many application fields. In particular Kohonen Networks, also known as SOM’s, belong to that kind of networks that do not require any explicit training stage. These unsupervised networks change their topology and their weights autonomously according to input data, by trying to replicate data morphology. Neurons are connected in a $n$-dimensional grid, where $n$ is the dimensionality of the input vectors, with a 4-way or 8-way connection scheme.

Interaction between couple of neurons will result in a modification of the weights in a specific area, determined by a Gaussian weight function, with the parameter $\sigma$ to determine the weight once the distance from the most activated neuron is known:

$$G(i, k) = e^{-\frac{|w_i - w_k|^2}{2\sigma^2}}$$

where $i$ and $k$ are two neurons and $w_i$, $w_k$ their weights. At the beginning, parameter $\sigma$ must be kept high so that the activation area is big. As the network progresses, the interaction area will get smaller so to influence a fewer number of neurons and to allow the algorithm to converge.

The activation rule is based on the Euclidean distance between input data and neuron weights. Let $i$ be the nearest neuron to the given input; then, for each neuron $k$, the weights of neurons are updated by using the following rule:

$$w_k(t) = w_k(t - 1) + \mu G(i, k)(T - w_k(t - 1))$$

where $\mu \in [0...1]$ is the training coefficient, $T \in [0...1]$ is the input, $N$ is the number of neurons, $(t - 1)$ is the previous iteration and $t$ is the actual iteration. Parameters $\mu$ and $\sigma$ are updated by multiplying them at each iteration by the learning rate $\alpha < 1$. During initialization, neuron weights are randomized in the range $[0...1]$.

#### 4.2. The SOM implementation

Features belonging to moving vehicles should have same speed vectors due to the rigid movement constraint. However, this constraint holds only within an affine camera model, when one can assume that perspective does not change the motion of the features belonging to the same object. Thanks to the SOM, we can relax that constraint allowing feature speeds to be recognized as belonging to the same object even though they are not identical. So we can track objects also in case that self-occlusion, strong perspective changes or deep field of view are present.

For each occluded object we use a 3-neuron SOM to clusterize features by speed. Our goal is to separate correctly tracked features from those undergoing a tracking error by using the SOM ability to match inputs topology. Tracking errors will be associated with the neuron showing an “anomalous” behaviour: consequently, given some rules (described in Section 4.3) we can remove features that are not well-tracked.

When we find more than one label in a single blob (i.e., during occlusions), for each group of features that share the same label we run a different SOM. Each network analyzes one object, searching for “anomalous” speeds derived from tracking errors; both $X$ and $Y$ components of the speed vector are used as SOM inputs. Due to the common motion of features we expect to find inputs (i.e. speeds) grouped in two clusters, a bigger one composed by inliers and a smaller
one made of outliers. Letting 3 neurons in a 2-D map, they will move until they cover the areas with the maximum density of inputs. Having 3 neurons and two high-density areas, two neurons will adapt their weights to cover the bigger cluster. Instead, the last neuron will move over the smaller one composed by inputs with incoherent speeds with respect to global object motion. We can tell if no tracking errors are present checking if all 3 neurons are close together, meaning the presence of only one cluster.

Speed is calculated for each feature by coordinate difference over the last two frames of the sequence, then normalized. This last operation is necessary because weights and inputs of our network must lie in the range [0...1]. To reduce sensitivity on slower features, we perform a non-linear normalization: given a minimum speed coefficient, if the maximum speed is under this threshold we use that coefficient to normalize, otherwise we refer all inputs to the fastest speed.

4.3. Outliers Rejection Rules

To detect outliers we use the norm of the distance from the axes center and the norm of the relative distances between the three pairs of neurons. Let us call $A$, $B$ and $C$ the neurons ordered by ascending distance from the origin, while $O$ is the center of the axes (Figure 1). According to each specific occlusion situation, we can identify three cases:

1. few features jump from a still object and attach to a moving one;
2. few features jump between two moving objects;
3. few features jump from a moving object to a still one (or to the background).

Now let us see how the feature speed displacements and the SOM perform in each case.

1. A large cluster of speed vectors near the axes center (where are located features of the still object with nearly zero speed) and a smaller cluster near the boundary of the map (corresponding to the fastest speed) must be present. Neurons are therefore divided in two groups (Figure 1, left): a couple ($A$ and $B$) near the origin and the third one ($C$) distant from the center of the axes. As a consequence, we consider $C$ as an outlier if $|CB| - |BA| > k$. That threshold gives a safety margin to detect the separation between the biggest group of features (inliers) from the smallest one (outliers). In Section 5, we explain how the value for the threshold $k$ is determined.

2. If few features jump between two moving objects, all neurons will be distant from the origin (Figure 1, center) so we calculate relative distances between each couple of neurons. If $\max(|AB|; |AC|; |BC|) > \max(k; 2 \min(|AB|; |AC|; |BC|))$ then there is an acceptable margin to consider the farthest neuron as an outlier. $k$ represents a minimum threshold to ensure separation between clusters, while the minimum distance between two out of the three neurons gives a measure of the density of the bigger cluster. A higher density helps to separate inliers from outliers.

3. Finally, if few features jump from a moving object to a still one because of a tracking error, the SOM will place one neuron ($A$) near to the origin whilst the remaining two ($B$ and $C$) are close together but far from the first one (Figure 1, right). To detect this particular SOM configuration we check if $|CB| - |BA| < -k$ and $|CO| - |AO| > |BO|$. The first inequality tests if neurons $B$ and $C$ are close together and far from $A$ at the same time. The second one tests the same condition but with respect to the origin, as to distinguish case 2 from this case.

In case 1 and 2, after proving the rules, we remove the label from the features associated with the farthest neuron. In case 3 features are deleted: in fact, features that stop moving while other (within the same object) have some speed are also likely to be weak corner points, which is why it is reasonable to remove them from the tracking process.

5. Experimental Results

As a first stage it is crucial to set the SOM parameters in order to achieve a good convergence and to maintain good overall performance. The number of iterations strictly depends on convergence speed and from $\sigma$ (see Section 4.1). According to these constraints, we set up the network with the values in Table 1. For each object involved in an occlusion, a network is initialized with the same parameters. The separation threshold $k$ referred in Section 4.3 has been experimentally set to $0.25$ ($\frac{1}{4}$ of the SOM weight range).
A higher value permits more tolerance on the rigid movement constraint, because features should have very different speeds to yield tracking errors. On the other hand a lower value for $k$ could lead to a high tracking error detection rate, because even a slight difference on speeds could yield the removal of the features associated with the farthest neuron.

To validate our algorithm we created a toy sequence T0 by using RC-cars, to produce rigid bodies interactions. T0 sequence (Figure 2) contains 28 occlusion cases, for a sum of 311 frames where cars overlap partially. Other properties of the sequence are in Table 2. All feature tracking errors are rejected by our algorithm, so that features are always correctly assigned to their respective objects. However, the tracker loses track of one car in two circumstances. The first occurs when all features are lost due to camera motion blur caused by high car speed. The second one occurs when all features are lost during occlusion because cars are too small due to the long distance from the camera. In this case the minimum feature density value chosen to elaborate the sequence has not permitted to extract new features in the region occupied by the cars since it was too small.

After the validation stage, the algorithm has been tested with portions of real world sequences shooted by the TMS of the city of Bologna (see Table 2 for details). In those sequences we want to measure traffic flows, counting vehicles as they pass across virtual lines drawn on the street surface. Each line owns two counters, triggered by vehicles that cross the line from left to right and vice versa. Counters are incremented when the centroid of a vehicle goes beyond one line between two subsequent frames. Figure 3 shows the T1 sequence (left) and how lines are placed (right). Figure 4 shows snapshots taken from T2 and T3 sequences with the virtual Lines. Due to the high number of vehicles, occlusions are very frequent (up to 5 different vehicles in the same blob) in all the sequences even between large and small objects like buses and motorcycles. These sequences show also shadows, a deep field of view and a low and uneven frame-rate: those characteristics increase the difficulty of tracking, because of the high number of tracking errors caused by the KLT algorithm. Besides correct tracking situations (HIT), errors can be classified as follows:

- **tracking miss (MISS)**: when an object changes its label during tracking, due to the loss of all features;
- **false positive (FP)**: when more than one label is assigned to the same object, after an erroneous blob split or a wrong occlusion management.

Results for traffic sequences are in Table 3. We now describe the most significant situations occurred in the T1 sequence, but the same considerations hold for T2 and T3 sequences as well.

- **Line A**: this line offers the best results in terms of successful tracking, because blobs overlap in small areas, thanks to the opportune camera tilt angle.
- **Line B**: the line is crossed by vehicles coming from the right. The FP detected are caused by the presence...
of three big buses, whose blobs merge with other vehicles. The SOM removes most of the tracking errors but some of them still remain because different objects had very similar speed vectors.

- **Line C**: most of the blobs entering the scene on the top of this line contain more than one vehicle. Anyway, merging and splitting of the blobs are correctly managed and no tracking errors are detected.

- **Line D**: although this line is the farthest one, results show that even small vehicles are tracked correctly.

- **Line E**: vehicles cross this line coming from three directions: along the entire sequence we can find small and fast objects as bigger and slower ones. Fast vehicles result in two MISS because high speed and uneven frame rate produce feature losses by the KLT algorithm. Traffic congestion for vehicles coming from the right side, as for line B, causes one FP detection after blob splitting. Occluded cars, moving in the same direction with similar speeds generate a scattered SOM which cannot reliably isolate tracking errors.

<table>
<thead>
<tr>
<th>Line</th>
<th>HIT (%)</th>
<th>MISS (%)</th>
<th>FP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 A</td>
<td>34</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T1 B</td>
<td>35</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>T1 C</td>
<td>27</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>T1 D</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T1 E</td>
<td>37</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: Results for T1, T2 and T3 traffic sequences

6. Conclusion and future work

In this paper we proposed a vehicle tracker based on KLT feature tracker which exploits a SOM to detect and remove feature tracking errors due to occlusions. In particular, the use of SOM allows to relax the rigid movement constraint and to track objects successfully also in case that self-occlusions, strong perspective changes or a deep field of view are present, thus increasing the overall robustness of the system. The tracking system has been tested on challenging sequences of urban traffic showing excellent performance in terms of tracking errors and number of vehicles correctly disoccluded. At present, our system has been working in real time in an experimental stage in the TMS of Bologna (Italy).

Acknowledgements

We would like to thank the Traffic Control Bureau of the Bologna Council in the person of Ing. Fabio Cartolano for having provided us with the traffic sequences used to test our prototype.

References


