TRACKING INTERACTING OBJECTS IN COMPLEX SITUATIONS BY USING CONTEXTUAL REASONING

Rosario Di Lascio[1*], Pasquale Foggia[2], Alessia Saggese[2] and Mario Vento[2†]
[1]*A.I.Tech s.r.l.*[‡], *Salerno, Italy*
[2]*Dipartimento di Ingegneria Elettronica e Ingegneria Informatica, University of Salerno, Salerno, Italy*

Keywords: Video Surveillance, Real-time Object Tracking.

Abstract: In this paper we propose a novel real-time tracking algorithm robust with respect to several common errors occurring in object detection systems, especially in the presence of total or partial occlusions. The algorithm takes into account the history of each object, whereas most other methods base their decisions on only the last few frames. More precisely, it associates each object with a state encoding the relevant information of its past history, that enable the most appropriate way of assigning an identity to the object on the basis of its current and past conditions. Thus, strategies that are more complex but also riskier are only applied when the algorithm is confident that is appropriate to do so. An experimental evaluation of the algorithm has been performed using the PETS2010 database, comparing the obtained performance with the results of the PETS 2010 contest participants.

1 INTRODUCTION

Object tracking algorithms are devoted to the task of reconstructing the object trajectories given the evidence collected from a video sequence. Although this task is apparently simple, several problems may hamper its performance: problems with the detection of the objects in each frame (objects missing or partially detected, spurious objects, objects split in parts), occlusions (a person is totally or partially covered by an element of the scene, or by another person), noise due to light changes or camera motion. Thus, many algorithms have been proposed in the literature for facing these problems, but none of them is both sufficiently reliable to operate in the complexity of a real world scenario, and sufficiently fast to work in real time.

The algorithms present in the literature can be roughly divided into two categories. In the first one, tracking is performed after an object detection phase: objects are detected in each frame using either some form of change detection (e.g. differences from a background model) or an a priori model of the objects. Algorithms in this category are usually faster, but they have to consider also the errors of the detection phase, such as spurious and missing objects, objects split into pieces, multiple objects merged into a single detected *blob*). As an example, the papers by (Seth and Jain, 1987), by (Rangarajan and Shah, 1991) and by (Intille et al., 1997) use a greedy algorithm that matches each object to its nearest neighbor, with constraints based on proximity. The first method assumes that the number of objects is constant, so it does not deal with object entries, exits and occlusions; the later methods add the ability to deal with entering or exiting objects and to recognize that an occlusion has occurred (without restoring the object identities after the occlusion).

The W^4 system by (Haritaoglu et al., 2000) uses the overlap of the areas as a criterion to find a correspondence between the objects at the current and at the previous frame. When this criterion selects multiple objects, the algorithm considers split or merge hypotheses to deal with detection errors or with occlusions. After an occlusion, an appearance model of the objects is used to reassign the original object identities. Also, when an object is seen for the first time, the algorithm waits for a fixed number of frames before assigning it an object identifier, in order to filter out spurious objects due to detection errors. The use of overlap works well with high frame rates and ob-

[*]This research has been partially supported by A.I.Tech s.r.l., a spin-off company of the University of Salerno (www.aitech-solutions.eu).

[†]IAPR Fellow.

[‡]A Spin-off company of the University of Salerno: www.aitech-solutions.eu.

104 Di Lascio R., Foggia P., Saggese A. and Vento M..
TRACKING INTERACTING OBJECTS IN COMPLEX SITUATIONS BY USING CONTEXTUAL REASONING.
DOI: 10.5220/0003819301040113
In *Proceedings of the International Conference on Computer Vision Theory and Applications* (VISAPP-2012), pages 104-113
ISBN: 978-989-8565-04-4
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

jects that are not very fast, but might fail in other conditions. The method proposed by (Chen et al., 2001) formulates the tracking problem as a bipartite graph matching, solving it by the Hungarian algorithm. It recognizes an occlusion, but is able to preserve the object identities only if the horizontal projection of the detected blob shows a separate mode.

In the second category, detection and tracking are performed at once, usually on the basis of an object model that is dynamically updated during the tracking. These methods are computationally more expensive, and often have problems with the initial definition of the object models, that in some cases has to be provided by hand. The paper by (Comaniciu et al., 2000) proposes the use of Mean Shift, a fast, iterative algorithm for finding the centroid of a probability distribution, for determining the most probable position of the tracking target. It requires a manual selection of the objects being tracked in the initial frame, and deals only with partial occlusions. (Tao et al., 2002) have proposed a method based on a layered representation of the scene, that is created and updated using a probabilistic framework. Their method is able to deal with occlusions, but is extremely computational expensive. The method by (Wu and Nevatia, 2005) tracks people in a crowded environment. However it uses an a priori model of a person, that is not extendable to other kind of objects. Several recent methods (Bazzani et al., 2010) have investigated the use of Particle Filters, that are a tool based on the approximate representation of a probability distribution using a finite set of samples, for solving the tracking problem in a Bayesian formulation. Particle Filters look very promising, since they make tractable a very general and flexible framework. However, the computational cost is still too high for real-time applications, especially with multiple occluding targets.

In this paper we propose a real-time tracking algorithm belonging to the first category; it assumes that an object detection based on background subtraction generates its input data. The algorithm is robust with respect to the errors generated by the object detection (spurious or missing objects, split objects) and is able to work with partial and total occlusions.

Most of the algorithms in the first category make their tracking decisions by comparing the evidence at the current frame with the objects known at the previous one; all the objects are dealt with in the same way, ignoring their past history that can give useful hints on how they should be tracked: for instance, for objects stable in the scene, information such as their appearance should be considered more reliable.

To exploit this idea, the algorithm adopts an object model based on a set of scenarios, in order to differently deal with objects depending on their recent history; the scenarios are implemented by Finite State Automata, each describing the different states of an object and the conditions triggering the transition to a different state. The state is used both to influence which processing steps are performed on each object, and to choose the most appropriate value for some of the parameters involved in the processing.

2 THE PROPOSED METHOD

Before starting the description of the algorithm, we need to introduce some terminology and notations.

A *blob* is a connected set of foreground pixels produced by a detection algorithm, which usually finds the foreground pixels by comparing the frame with a background model; then the foreground pixels are filtered to remove noise and other artifacts (e.g. shadows); finally, the foreground pixels are partitioned into connected components, which are the blobs. The tracking algorithm receives in input the set of blobs detected at each frame. We assume that the detection phase uses a dynamic background model dealing with lighting changes; noise reduction, shadow and small blob removal are further carried out. See details in (Conte et al., 2010).

An *object* is any real-world entity the system is interested in tracking. Each object has an *object model*, containing such information as the object class (e.g. a person or a vehicle), state (see subsection 2.1), size, position, trajectory and appearance (see subsection 2.4). A *group object* corresponds to multiple real-world entities tracked together; if a group is formed during the tracking (i.e. it does not enter the scene as a group), its object model mantains a reference to the models of the individual objects of the group.

The task of the tracking algorithm is to associate each blob to the right object, in such a way as to preserve the identity of real-world objects across the video sequence; in the process the algorithm must also create new object models or update the existing ones as necessary.

In real cases, the detection phase produces some common errors:

- *Spurious Blobs*, i.e. blobs not corresponding to any object; they can be caused by lighting changes, movements of the camera or of the background, and other transient changes that the detection algorithm was not able to filter out;
- *Ghost Blobs*, i.e. blobs appearing where there was an object previously considered as part of the background, that has moved away (e.g. a parked car that starts moving);

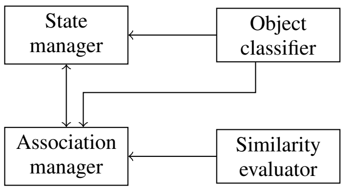


Figure 1: An overview of the tracking system.

```
procedure Tracking(obj_models, blobs)
  Classify(current_blobs)
  S := ComputeSimilarity(obj_models,
                         blobs)
  FindAssociations(obj_models, blobs, S)
  UpdateModels(obj_models, blobs)
  UpdateState(obj_models)
end procedure
```

Figure 2: The structure of the tracking algorithm.

- *Missing Blobs*, i.e. undetected objects, for instance as too similar to the background behind them;
- *Split Blobs*, i.e. objects divided into multiple blobs.

In addition the algorithm must also handle partial or total occlusions, ensuring that object identities are not lost across the occlusion.

A key idea behind the proposed algorithm is to base the decisions regarding an object not only on its current conditions, but also on its past history; in this way spurious observations can be easily ignored, and the decisions can be based on stable properties of the object. To this aim, the object history is encoded using a *state*, belonging to a finite set of possible values. The transitions between states are explicitly described through a Finite State Automaton, and are triggered by such events as the permanence of the object in the scene, its disappearance, the outcome of the object classification and the participation to an occlusion.

The object state influences decisions taken by the *association management* module, which establishes a correspondence between objects and blobs, solving split-merge events and performing occlusion reasoning. Association management is mainly based on a similarity measure between objects and blobs, considering position, size and appearance.

Figure 1 shows the modules composing the tracking system, and their interdependencies, while Figure 2 shows an outline of the tracking algorithm. In the following subsections, more details are provided for each module of the system.

2.1 Object State Management

The state manager is based on the Finite State Automaton $\mathcal{A}$ depicted in Figure 3, that can be formally

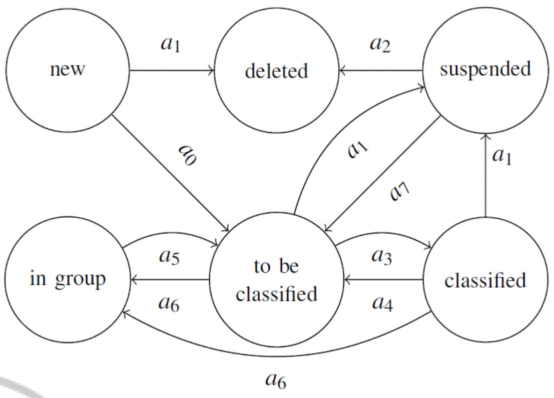


Figure 3: The state diagram of the object state manager.

defined as:

$$\mathcal{A} = \langle S, \Sigma, \delta, s_0, F \rangle \quad (1)$$

where $S = \{s_0, \ldots, s_m\}$ is the set of the states; $\Sigma = \{a_0, \ldots, a_m\}$ is the set of the transition conditions, i.e. the conditions that may determine a state change; $\delta : S \times \Sigma \rightarrow S$ is the state-transition function; $s_0 \in S$ is the initial state and $F \subset S$ is is the set of final states.

The proposed Finite State Automaton states and transitions are shown in table 1. In particular, the set of states S is shown in table 1.a; we choose s_0 as initial state, since each object enters the scene by appearing either at the edge or at a known entry region (e.g. a doorway). Furthermore we choose s_5 as final state, since each object necessarily has to leave the scene. The set Σ of transition conditions and the state-transition function δ are shown respectively in Table 1.b and 1.c.

The meaning of the states and the conditions triggering the transitions are detailed below:

- *new* (s_0): the object has been just created and is located at the borders of the frame; if it enters completely, and so does not touch the frame borders (a_0), it becomes *to be classified*; otherwise, if it leaves the scene (a_1), it immediately becomes *deleted*;
- *to be classified* (s_1): the object is completely within the scene, but its class is not yet considered reliable; if the classifier assigns the same class for at least two frames (a_3), it becomes *classified*; if the association manager detects that the object has joined a group (a_6), it becomes *in group*; if the object disappears (a_1), it becomes *suspended*;
- *classified* (s_2): the object is stable and reliably classified; if the classifier assigns a different class (a_4), it becomes *to be classified*; if the association manager detects that the object has joined a group (a_6), it becomes *in group*; if the object disappears (a_1), it becomes *suspended*;

Table 1: The Finite State Automaton. (a) The set S of the states. (b) The set Σ of the transition conditions. (c) The state-transition function δ; for entries shown as '-', the automaton remains in the current state.

(a)

Id	Description
s_0	new object
s_1	to be classified object
s_2	classified object
s_3	suspended object
s_4	in group object
s_5	deleted object

(b)

Id	Description
a_0	obj enters completely within the scene
a_1	obj disappears from the scene
a_2	obj does not reappear in the scene for a time T_d
a_3	obj classification changes
a_4	obj classification remains the same for two frames
a_5	obj leaves the group
a_6	obj occludes with one or more objects
a_7	obj reappears inside the scene

(c)

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
s_0	s_1	s_5	-	-	-	-	-	-
s_1	-	s_3	-	s_2	-	-	s_4	-
s_2	-	s_3	-	-	s_1	-	s_4	-
s_3	-	-	s_5	-	-	-	-	s_1
s_4	-	-	-	-	-	s_1	-	-
s_5	-	-	-	-	-	-	-	-

- *in group* (s_4): the object is part of a group, and is no more tracked individually; its object model is preserved to be used when the object will leave the group; if the association manager detects that the object has left the group (a_5), it becomes *to be classified*;
- *suspended* (s_3): the object is not visible, either because it is completely occluded by a background element, or because it has left the scene; if the object gets visible again (a_7), it becomes *to be classified*; if the object remains suspended for more than a time threshold T_d (a_2), it becomes *deleted*; currently we use $T_d = 1$ sec;
- *deleted* (s_5): the object is not being tracked anymore; its object model can be discarded.

The use of the *new* state allows the algorithm to quickly discard spurious objects due to detection artifacts, since they usually do not persist long enough to become *to be classified*. On the other hand, the *suspended* state avoids that an object is forgotten too soon when it momentarily disappears because of a detection miss, an occlusion with the background or a temporary exit from the scene. The *in group* state has the purpose of keeping the object model even when the object cannot be tracked individually, as long as the algoritm knows it is included in a group. Finally, the distinction between *classified* and *to be classified* objects is used by the association manager when reasoning about split objects and group formation.

Figure 4 shows in a very simple example how the object state management works.

2.2 Object Classification

The tracking system needs an object classifier to determine if a blob corresponds to a group, an individual object, or an object part. Currently we have applied our system to people tracking, so we have only two classes of individual objects: person and baggage.

For these classes, the width and the height are a sufficiently discriminant feature vector; if the algorithm should be applied to a problem with more classes, other features could be added, for example based on the shape of the blob. In order to obtain the actual width and height of the object, removing the scaling introduced by the perspective, we perform an Inverse Perspective Mapping (IPM) based on camera calibration data.

IPM is a geometrical transformation technique that reconstructs the 3D position of an object from its 2D position in the image plane (Muad et al., 2004). Once the 3D position is known, the actual size of the object can be easily computed from its apparent size.

The classifier we have implemented is a simple Nearest Neighbor classifier, that has a reference set of a few objects for each class (including groups of people). More in detail, we have a set of reference vectors, $R = \{r_1, \dots, r_k\}$ where each r_i a feature vector (w_i, h_i) containing the width and the height of the reference object; each r_i is associated to a known class, denoted as $\text{class}(r_i)$. The blob that has to be classified is represented by a feature vector $v = (w_v, h_v)$. Then the class is determined as follows:

$$j = \arg\min_i \|v - r_i\| \tag{2}$$

$$\text{class}(v) = \text{class}(r_j) \tag{3}$$

where $\|.\|$ denotes the Euclidean norm, and j is the index of the reference vector that is nearest to v. The value of $\|v - r_j\|$ can be used to check if the classification is reliable (because the blob is very similar to one of the reference objects) or not. If:

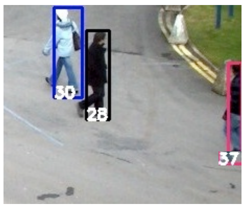
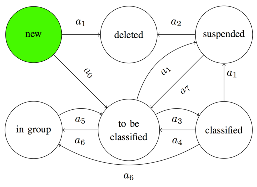
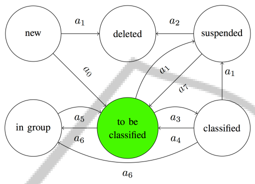
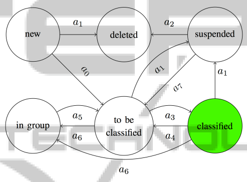
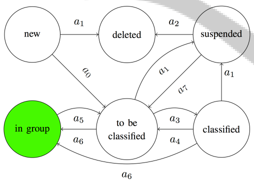
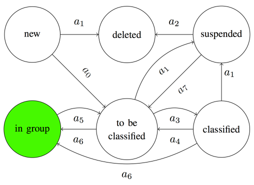
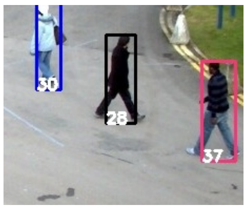
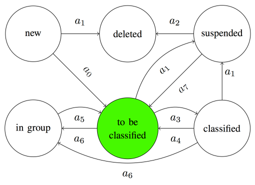
		The object 37 enters the scene as a *new* object.
		The object 37 becomes *to be classified*.
		When its classification as a person object becomes reliable, the objects becomes *classified*.
		After a few frames, the association manager identifies an occlusion and the object 37 becomes *in group*.
		The object 37 remains inside the group, so it does not change its state.
		The association manager identifies a split, but it does not solve it since it is associated to a group object. In such a way, the object exits the group and it becomes *to be classified* again.

Figure 4: Each object is associated to a Finite State Automaton that aims to manage the history of an object. We focus attention on the object identified by the number 37 since it is entering the scene.

$$\|v - r_j\| > \theta_c \tag{4}$$

where θ_c is a classification threshold, the algorithm assumes that the blob is likely the result of a split in the detection phase.

At this point one can note that object evolution is not dependent on its class (e.g. group or individual

```
procedure TrackingAlgorithm (obj_models, blobs)
 AssociationStableObjs (obj_models, blobs)
 pending_blobs := SearchPendingBlobs(blobs)
 AssociationInstableObjs (obj_models, pending_blobs)
 unassociated_blobs := SearchPendingBlobs(pending_blobs)
 CreateObjFromPendingBoxes(obj_models,unassociated_blobs)
 UpdateObjectsState(obj_models)
end procedure

procedure AssociationInstableObjs (obj_models, blobs)
 sim_mat := ObjInstableSimilarityMatrix(obj_models, blobs)
 for all obj in obj_models:
  (best_boxes, best_objs) := BestAssociation(sim_mat)
 end
end procedure

procedure AssociationStableObjs (obj_models, blobs)
 sim_mat := ObjStableSimilarityMatrix(obj_models, blobs)
 for all obj in obj_models:
  (best_boxes, best_objs) := BestAssociation(sim_mat)
  SolveMerge(best_boxes, best_objs)
  SolveSplit(best_boxes, best_objs)
 end
end procedure
```

Figure 5: The structure of the algorithm for stable and instable objects associations.

object), but only on its actual state. As a matter of fact, only object information is related to object class, while object state only determines the reliability of such information. In particular, for individual objects we have information about appearance and shape: we consider the area and the perimeter of an object, its color histograms and its real dimensions, that is width and height, both obtained using an Inverse Perspective Mapping. Moreover we have information about the observed and predicted position of the object centroid. The predicted position is obtained using an extended 2D Position-Velocity (PV) Kalman Filter, whose state vector is:

$$\xi = \left[x_c, y_c, w, h, \dot{x}_c, \dot{y}_c, \dot{w}, \dot{h}\right] \tag{5}$$

where (x_c, y_c) is the centroid of the object, w and h are the width and the height of the object minimum bounding box in pixels, $(\dot{x}_c, \dot{y}_c)$ and $(\dot{w}, \dot{h})$ are respectively the velocity of the object centroid and the derivative of the minimum bounding box size. It is worth noting that such a PV Kalman Filter is very effective when the object motion is linear and the noise has a Gaussian distribution.

Group objects contain also information about occluded objects. In this way the system can continue to track the *in group* objects when they leave the group.

2.3 Association Management

The task of the association manager is to establish a correspondence between the objects and the blobs detected at the current frame. This correspondence can be represented as a matrix $T = \{t_{ij}\}$ defined as:

$$t_{ij} = \begin{cases} 0 & \text{if obj. } o_i \text{ is not associated to blob } b_j \\ 1 & \text{if } o_i \text{ is associated to } b_j \end{cases} \tag{6}$$

In order to perform this task, the association manager may introduce new object models, and it may request an update of the existing ones.

In simple cases, there is a one-to-one correspondence between an object and a blob; in such cases, at most one element has the value 1 in each row and in each column of T:

$$\sum_i t_{ij} \leq 1 \, \forall i; \sum_j t_{ij} \leq 1 \, \forall j \tag{7}$$

However, the association manager has to consider more complex associations (one-to-many, many-to-one, and even many-to-many) in order to deal with occlusions and with split blobs.

The algorithm operates in two distinct phases, as shown in Figure 5: in the first one it finds the correspondence for stable objects (i.e. objects in the *to be classified* or *classified* state); in the second phase it tries to assign the remaining blobs to objects in the *new* state, possibly creating such objects if necessary. The motivation for this distinction is that split-merge and occlusion reasoning is only performed for stable objects, since for new objects the system would not have enough information to do it in a reliable way.

The algorithm for the association of stable objects is shown in Figure 5. It is a greedy algorithm, based on the use of a similarity matrix. Rows and columns of the similarity matrix are respectively used to represent the objects and the blobs. In this way each element s_{ij} of the matrix represents a similarity measure between the blob b_j and the object o_i. The construction of the similarity matrix and of the similarity index is described in detail in subsection 2.4.

The algorithm starts by choosing the maximum element of the matrix; if its value is above a given threshold τ, the algorithm records the corresponding association:

$$(k, l) = \arg\max_{ij} s_{ij} \tag{8}$$

$$t_{k,l} = 1 \text{ if } s_{kl} > \tau \tag{9}$$

where k and l represent the indices of the maximum element of the similarity matrix.

Then the algorithm checks if the blob b_l of this association has other objects that are close to it and are not similar enough to different blobs, as an evidence that an occlusion is starting, or that a detached object part (or a baggage) is becoming attached to the object; this condition can be formulated as:

$$\exists o_m \neq o_k : s_{ml} > \tau \wedge s_{ml} = \max_j s_{mj} \tag{10}$$

The association manager uses the current state and the output of the classifier to discriminate between the two kinds of event, and in the first case it creates a group object and links it with the individual objects forming the occlusion.

At this point the algorithm verifies if the object of the selected association o_k has other blobs that are close to it and are not similar enough to different objects; this may be caused by either the end of an occlusion, or by a split blob; more formally:

$$\exists b_n \neq b_l : s_{kn} > \tau \wedge s_{kn} = \max_i s_{in} \tag{11}$$

Again, the association manager uses the current state and the classifier outputs to recognize the correct event; in the case of an ending occlusion, the algorithm uses the individual object models linked to the group object to reassign the correct identity to the objects leaving the group, changing their state from *in group* to *to be classified*.

Finally, the algorithm removes from the similarity matrix all the rows and columns corresponding to objects and blobs it has used, and repeats the choice of the maximum element in the matrix. If no element is above the threshold τ, all the remaining unassigned objects are put in the *suspended* state and the first phase terminates.

The second phase is shown in Figure 5. It follows a similar scheme, except that it considers only the objects in the *new* state, and does not perform the checks for merges, splits, starting and ending occlusions. Moreover, the similarity matrix is built using less features than in the first phase since we have experimentally verified that only the position information (see section 2.4) is sufficiently reliable for such objects. At the end of this phase, any remaining unassigned blobs are used to create new object models, initialized to the *new* state.

2.4 Similarity Evaluation

As already mentioned, the similarity matrix is used to match one or more blobs with one or more objects. In order to measure the similarity between an object o_i and a blob b_j, the tracking system uses an index based on three kinds of information: the position, the shape and the appearance:

$$s_{ij} = \sqrt{\frac{\alpha_p \cdot (s_{ij}^p)^2 + \alpha_s \cdot (s_{ij}^s)^2 + \alpha_a \cdot (s_{ij}^a)^2}{\alpha_p + \alpha_s + \alpha_a}} \tag{12}$$

As described below, s_{ij} values identify similarity metrics and α values are weights chosen according to the state of the object and the association management phase. In particular:

- s_{ij}^p is the position similarity index, that is the distance between the estimated centroid of an object o_i and the centroid of a blob b_j;
- s_{ij}^s is the shape similarity index between an object o_i and a blob b_j;
- s_{ij}^a is the appearance similarity index between an object o_i and a blob b_j, based on color histograms;
- α_p, α_s and α_a are the weights of position, shape and appearance similarity index respectively;

All α values have been chosen by experimentation over a training set. Namely, in the first phase, selected values for objects in the *to be classified* and *classified* state are $\alpha_p = \alpha_s = \alpha_a = 1$ while for objects in the *in group* state selected values are $\alpha_s = \alpha_a = 1$; $\alpha_p = 0$ since in this context shape and appearance similarity perform better than position one. Finally, in the second phase that evaluates *new* objects, we choose to consider the only reliable feature, that is the position. Thus selected α values are $\alpha_s = \alpha_a = 0$; $\alpha_p = 1$.

For the position, as we have already seen, the system uses a Kalman filter, based on a uniform velocity model, to predict the coordinates of the object centroid at the current frame. The predicted coordinates are compared with the center of the blob, using Euclidean distance, obtaining for each object o_i and each blob b_j the distance d_{ij}. The position similarity index is then computed as:

$$s_{ij}^p = 1 - d_{ij}/d_{\max} \tag{13}$$

where $d_{\max}$ is a normalization factor depending on the maximum velocity of objects representing the maximum displacement of an object between two frames.

For characterizing the shape similarity, the system uses the real height and the area of the blob and of the object model; in particular if we denote as Δh_{ij} the relative height difference between o_i and b_j, and as ΔA_{ij} the relative area difference, the considered shape similarity index is:

$$s_{ij}^s = 1 - \sqrt{\frac{(\Delta A_{ij})^2 + (\Delta h_{ij})^2}{2}} \tag{14}$$

Finally, as a representation of the appearance we have used the color histograms computed separately for the upper half and for the lower half of the object or blob (*Image Partitioning*). We have experimented with several criteria for comparing the histograms, and we have found that the most effective value is the χ^2 distance:

$$q_{ij} = \frac{1}{M} \sum_k \frac{\left(h_i^o(k) - h_j^b(k)\right)^2}{h_i^o(k) + h_j^b(k)} \tag{15}$$

where index k iterates over the bins of the histogram, h_i^o is the histogram of object o_i, h_j^b is the histogram of blob b_j, and M is the number of bins. The appearance similarity index is:

$$s_{ij}^a = 1 - \sqrt{\frac{\left(1 - q_{ij}^{up}\right)^2 + \left(1 - q_{ij}^{low}\right)^2}{2}}. \tag{16}$$

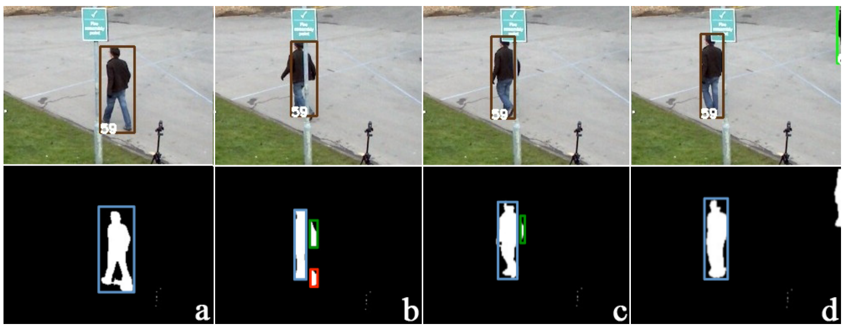


Figure 6: The output of the method on a PETS2010 sequence containing a split, caused by the presence of a pole in the scene. The first row shows the result of the method, while the second one the list of the blobs, which is the input of the tracking phase. Note that the object identified by 59 is correctly tracked, not withstanding the split in frames (b) and (c).

Table 2: The real number of occurrences of the objects in the scenes and the number of id switches.

View	Objs	Objs Occurrences	Correctly Identified Objs	Id Switch
1	22	4840	4592 (95%)	92 (3%)
3	21	6377	4925 (77%)	294 (5%)
4	22	6076	4440 (73%)	158 (3%)
5	28	2722	2344 (86%)	76 (3%)
6	32	3141	2621 (84%)	156 (5%)
7	29	4578	3225 (70%)	147 (3%)
8	26	4310	3406 (79%)	107 (2%)

where q_{ij}^{up} is the value of q_{ij} computed using only the upper half of the object/blob, and q_{ij}^{low} is the value computed using only the lower half.

3 EXPERIMENTAL VALIDATION

In order to assess the performance of the method with respect to the state of the art, we have used the publicly available PETS 2010 database (13th IEEE Int. Workshop on Performance Evaluation of Tracking and Surveillance, 2010), currently used by many research papers. It is made of seven videos captured in a real-world environment, containing several occlusions between a person and an object, two persons or among several persons. We have computed in each view the maximum velocity of the objects, from which we have derived the optimal values of the d_{max} parameter of equation 13, that are $d_{max} = 100$ for views 1, 3 and 4 and $d_{max} = 150$ for view 5, 6, 7 and 8.

Figures 6 and 7 show two excerpts, respectively containing a split pattern and a complex occlusion pattern among three persons; as it can be seen, in both the situations the system preserves the object identities across the occlusion and the split. A quantitative evaluation has been carried out using the performance indexing proposed in the PETS 2010 contest (Ellis and Ferryman, 2010). In particular, we have used the following indices: the *Average Tracking Accuracy* (ATA), the *Multiple Object Tracking Accuracy* (MOTA) and the *Multiple Object Tracking Precision* (MOTP). In the following we introduce some notations useful to formally define them.

Let $G_i^{(t)}$ and $D_i^{(t)}$ be the ith ground truth object and the detected one in frame t; $N_G^{(t)}$ and $N_D^{(t)}$ denote the number of ground truth objects and detected ones in frame t, respectively, while N_G and N_D denote the number of ground truth objects and unique detected ones in the given sequences. N_{frames} is the number of frames in the sequences. Finally, N_{mapped} refers to the mapped system output objects over an entire reference track, taking into account splits and merges and and $N_{mapped}^{(t)}$ refers to the number of mapped objects in the tth frame.

ATA is a spatiotemporal measure that penalizes fragmentations in spatiotemporal dimensions while accounting for the number of objects detected and tracked, missed objects, and false positives. ATA is defined in terms of Sequence Track Detection Accuracy STDA:

$$STDA = \sum_{i=1}^{N_{mapped}} \frac{\sum_{t=1}^{N_{frames}} \frac{|G_i^{(t)} \cap D_i^{(t)}|}{|G_i^{(t)} \cup D_i^{(t)}|}}{N_{G_i \cup D_i \neq 0}}. \quad (17)$$

La latter measures the overlap in the spatiotemporal dimension of the detected object over the ground truth, taking a maximum value of N_G. The ATA is defined as the STDA per object:

$$ATA = \frac{STDA}{\left[\frac{N_G+N_D}{2}\right]}. \quad (18)$$

As already mentioned, MOTA is an accuracy score computing the number of missed detects, false positives and switches in the system output track for a

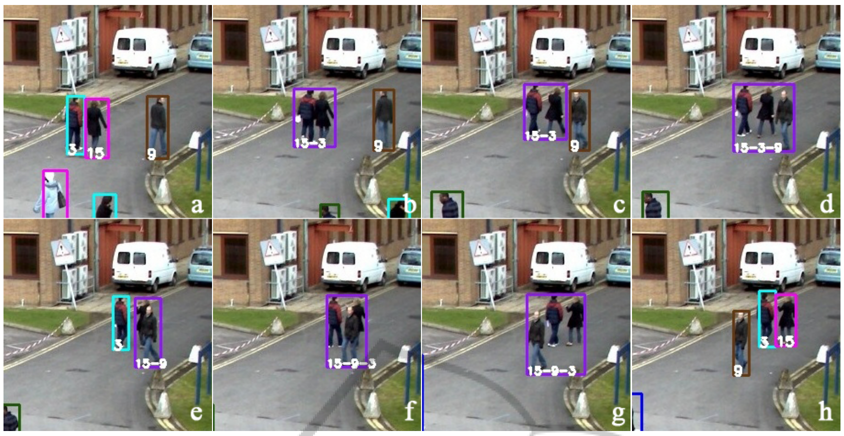


Figure 7: The output of the proposed method on a PETS2010 sequence containing an occlusion. Note how the object 9 is correctly tracked inside the different groups although it quickly changes its direction in the frame (c).

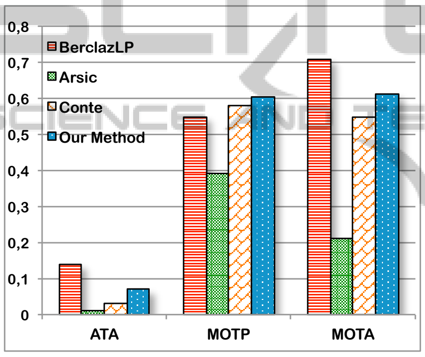


Figure 8: Performance of the method compared with the PETS 2010 contest participants on all the views.

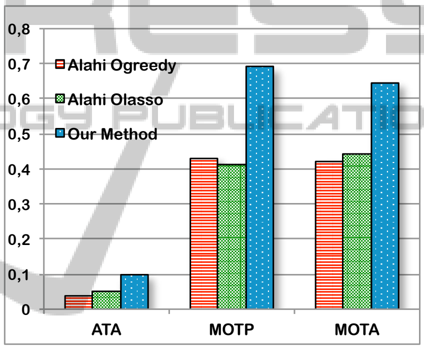


Figure 9: Performance of the method compared with the PETS 2010 contest participants on views 1, 5, 6 and 8.

given reference ground truth track. It is defined as:

$$MOTA = 1 - \frac{\sum_{t=1}^{N_{frames}} \left(c_m \cdot m_t + c_f \cdot fp_t + c_s \cdot is_t \right)}{\sum_{t=1}^{N_{frames}} N_G^{(t)}}, \tag{19}$$

where m_t is the number of misses, fp_t is the number of false positives, and is_t is the number of ID mismatches in frame t considering the mapping in frame $(t-1)$; c values are weights chosen as follow: $c_m = c_f = 1; c_s = log_{10}$.

Finally, MOTP is a precision score calculating the spatiotemporal overlap between the reference tracks and the system output tracks:

$$MOTP = \frac{\sum_{i=1}^{N_{mapped}} \sum_{t=1}^{N_{frames}^{(t)}} \frac{|G_i^{(t)} \cap D_i^{(t)}|}{|G_i^{(t)} \cup D_i^{(t)}|}}{\sum_{t=1}^{N_{frames}} N_{mapped}^{(t)}}. \tag{20}$$

Before analysing the performance of the method, let us point out some properties of the considered views. The first view presents interactions among two or three objects; the only difficulty is due to the presence of the pole and of the sign hanged on it, which causes a lot of splits. Note that the proposed method proves to be particularly robust with respect to the split situations on this view.

Views 3 and 4 are the most complex; in particular, view 3 is characterized by the presence of a large tree (about one-third of the scene), occluding a lot of individual or group objects. The situation is further complicated by the complexity of interactions among the objects, which involves in the average $2-5$ objects for view 3 and $2-6$ for view 4. Another problem in view 4 is the presence of a white-orange ribbon, continuously moving because of the wind. Such situation causes a lot of problems also in the detection phase.

The problem of the moving ribbon is also present in views 5, 6, 7 and 8, even if it is less visible. We can note that the performance obtained in views 6 and 7 is generally lower than that obtained on other sequences; this is related to more complex interactions between the tracked objects, having a very high number of occlusions associated to objects that are entering the scene (unstable objects).

It is worth noting that the method, during an occlusion, does not attempt to find the exact position of an object inside a group; it continues to track the group as a whole, using the Kalman filter for obtaining a prevision of the position of each object inside the group itself; this choice obviously causes a degradation of the performance if it is measured using indices defined assuming that objects are always tracked individually.

Figures 8 and 9 show the performance of the method, compared with the participants on the PETS 2010 competition. It is worth noting that the results presented to the PETS 2010 by other competitors in some cases only refer to a subset of the views (Views 1, 5, 6 and 8). For such reason, in order to have a proper comparison with these methods, we present the experimental results computed both over all the views and over the same subset of views as these methods. In particular, the results in Figures 8 refer to all the views of the database, while Figure 9 only refers to views 1, 5, 6 and 8. We can note that in the first comparison our method outperforms the others on the precision index (MOTP), while in the second one it clearly outperforms all the other participants of the context on these views on all the indices. Table 2 presents for each view a detail of the performance of our algorithm. As for the computational cost, the system runs at 16 milliseconds per frame on 4CIF images, using an Intel Xeon processor at 3.0GHz.

4 CONCLUSIONS

In this paper we have presented a real-time tracking algorithm able to overcome many of the problems of the object detection phase, as well as total or partial occlusions. It has been experimentally validated on a public database, showing a significant performance improvement over the participants to an international competition.

REFERENCES

13th IEEE Int. Workshop on Performance Evaluation of Tracking and Surveillance (2010). The PETS 2010 benchmark data. http://www.cvg.rdg.ac.uk/PETS2010/.

Bazzani, L., Cristani, M., and Murino, V. (2010). Collaborative particle filters for group tracking. In *IEEE Int. Conf. on Image Processing*, pages 837–840.

Chen, H.-T., Lin, H.-H., and Liu, T.-L. (2001). Multi-object tracking using dynamical graph matching. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 210–217.

Comaniciu, D., Ramesh, V., and Meer, P. (2000). Real-time tracking of non-rigid objects using mean shift. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 142–149.

Conte, D., Foggia, P., Percannella, G., Tufano, F., and Vento, M. (2010). An experimental evaluation of foreground detection algorithms in real scenes. In *EURASIP Journal on Advances in Signal Processing*, volume 11.

Ellis, A. and Ferryman, J. (2010). PETS2010 and PETS2009 evaluation of results using individual ground truthed single views. In *IEEE Int. Conf. on Advanced Video and Signal Based Surveillance*, pages 135–142.

Haritaoglu, I., Harwood, D., and David, L. S. (2000). W4: Real-time surveillance of people and their activities. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8):809–830.

Intille, S. S., Davis, J. W., and Bobick, A. F. (1997). Real-time closed-world tracking. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 697–703.

Muad, A., Hussain, A., Samad, S., Mustaffa, M., and Majlis, B. (2004). Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system. In *TENCON 2004. 2004 IEEE Region 10 Conference*, volume A, pages 207 – 210 Vol. 1.

Rangarajan, K. and Shah, M. (1991). Establishing motion correspondence. *CVGIP: Image Understanding*, 54(1):56 – 73.

Seth, I. and Jain, R. (1987). Finding trajectories of feature points in a monocular image sequence. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 9(1):56–73.

Tao, H., Sawhney, H. S., and Kumar, R. (2002). Object tracking with bayesian estimation of dynamic layer representations. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(1):75–89.

Wu, B. and Nevatia, R. (2005). Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In *Tenth IEEE Int. Conf. on Computer Vision*, volume 1, pages 90–97.