Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2006-027 April 25, 2006

Detecting and tracking multiple interacting objects without class-specific models

Biswajit Bose, Xiaogang Wang, and Eric Grimson

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Detecting and tracking multiple interacting objects without class-specific models

Abstract

We propose a framework for detecting and tracking multiple interacting objects from a single, static, uncalibrated camera. The number of objects is variable and unknown, and object-class-specific models are not available. We use background subtraction results as measurements for object detection and tracking. Given these constraints, the main challenge is to associate pixel measurements with (possibly interacting) object targets. We first track clusters of pixels, and note when they merge or split. We then build an inference graph, representing relations between the tracked clusters. Using this graph and a generic object model based on spatial connectedness and coherent motion, we label the tracked clusters as whole objects, fragments of objects or groups of interacting objects. The outputs of our algorithm are entire tracks of objects, which may include corresponding tracks from groups of objects during interactions. Experimental results on multiple video sequences are shown.

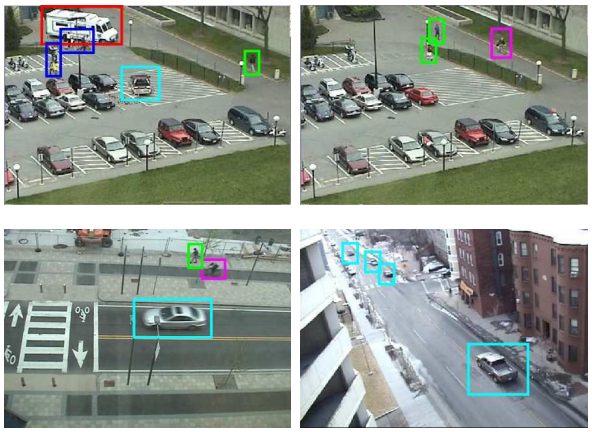


Figure 1. Our goal is to simultaneously track different types of objects, *e.g.* cars (marked in cyan), trucks (red), single persons (green), group of people (blue), and bicycles (magenta). Objects will be occluded when they interact. There is also signifi cant variation in size and appearance of objects because of projective distortion and pose change.

1. Introduction

In many unrestricted environments, where there are a large number of object classes of interest, it is not feasible to use class-specific appearance and dynamics models for each object class. For instance, in urban scenes, typical classes of moving objects include pedestrians, cars, motorbikes, vans, trucks, bicycles and pets. Building class-specific models requires either training detectors on sufficient labeled examples from each class or designing all these models by hand. It is especially difficult to make these appearance models robust to viewpoint and scale changes (see Figure 1). Further, for applications such as video surveillance and traffic analysis, the most interesting objects are sometimes ones that have not been seen before.

In the absence of class-specific models, objects can still be detected using a generic object model. For instance, moving objects can be detected by clustering foreground pixels after background subtraction [22]. This model is generic in the sense that it cannot distinguish between different object classes. It can however discriminate between moving objects of interest on the one hand and the stationary background (and randomly appearing noise) on the other.

We consider the use of generic object models for tracking multiple interacting objects (*i.e.*, objects whose observed trajectories or appearances are influenced by each other). An example of an interaction event is when one object occludes another with respect to a given camera view (for example, two cars crossing each other, where the camera is not looking top-down). Two people approaching each other, stopping to meet and then walking together for some time, is a more complicated example of interaction. Even if class-specific object models are known, performing occlusion reasoning is not easy.

An ambiguity arises when using generic object models to track multiple interacting objects in environments where the number of objects is varying and unknown (*i.e.*, objects may enter or exit at any time). This difficulty is best illustrated with a background-subtraction example. Here, a single object may get split into multiple foreground blobs (if the object and background have similar colors). At the same time, a group of interacting objects may get merged into a single foreground blob. These two cases are very hard to distinguish from the case of a single object pro-

Figure 2. We show three (zoomed-in) foreground images here, to illustrate the fragment-object-group ambiguity. In the first two frames, there are 1 and 3 foreground blobs respectively, so it is hard to tell that there are actually 2 objects throughout. In particular, in the middle frame, the gap between the two fragments of the object on the left is similar to the gap between the two objects. Note that object size is unknown. By tracking these blobs further and using coherent motion properties, the decision becomes easier.

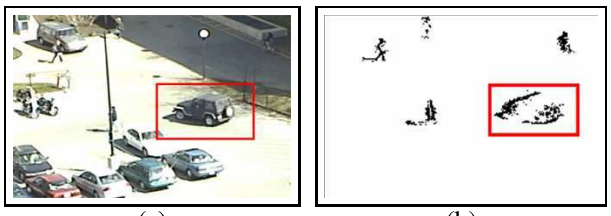
ducing a single foreground blob, since we simultaneously need to estimate the number of objects and the association of foreground pixels with objects. The best solution is to accumulate further evidence from multiple frames and use temporal coherence properties (see Figure 2). Existing generic-object tracking systems often incorrectly associate a single object with measurements from a group of objects, or track fragments of an object as independent objects [22]. Further, many trackers do not maintain identities of object tracks before and after interactions [6, 15]. Preserving identity information throughout an object's existence is essential for higher-level tasks such as activity recognition.

We propose a unified framework for tracking objects and groups of objects under the following constraints:

1. no class-specific object models,
2. a varying and unknown number of possibly interacting objects,
3. a single, static, uncalibrated camera, and
4. arbitrary viewing angle of camera and 3D pose of objects.

The motivation for our work is to be able to track a diverse collection of objects in an unrestricted, *open-world* environment, without using any scene-specific training.

Given the output of a background-subtraction system, we track clusters of foreground pixels till they merge or split. Each such tracked cluster (which we call a *target-set*) may be a fragment of an object, a whole object or a group of objects. To be able to label these target-sets, we define a generic object model based on spatial locality and coherent motion of constituent parts. We use this definition, along with a graph-based representation of merges and splits, to infer the number of objects and groups, as well as their association with measurements. By tracking groups as distinct entities whenever they occur, we avoid performing occlusion reasoning. Instead, we simply associate object tracks that merged to form a group with object tracks that split from the group. In this way, we maintain tracks of all moving objects with correct identities through multiple interaction events.



(a) (b)

Figure 3. (a) An example scene. (b) The foreground image for this scene. It is hard to distinguish objects from fragments using size-based criteria. Here the silhouette of a car has big fragments (comparable to people in size), with a large gap between fragments.

1.1. Related work

Many model-based multi-object tracking methods have been developed for detecting and tracking a single object-class of interest, such as tracking multiple humans ([8, 11, 13, 14, 20, 21, 24, 25]). The object model used by these methods might be an appearance model describing various parts of the object in the image, or simply a bounding box of specified size that can be applied to foreground data. Such methods often further assume that multiple cameras [13], or a single calibrated camera [8], are available, thus allowing for geometric reasoning in three dimensions when objects interact. The object model may only apply to a single view (such as a side view or a front view of a vehicle). Sometimes, a single camera is set up with a top-down view of the scene—in surveillance settings, for instance—so that inter-object occlusion is minimized. These methods can handle a varying number of objects of the same class (satisfying their restricted viewing assumptions). Of course, these methods are not suitable for tracking objects of multiple classes that occur in varying poses.

The multi-target tracking problem studied in the signal processing community, for applications such as radar tracking, is similar to our problem in the sense that class-specific object observation or dynamic models are not available [1, 3]. However, the target is often treated as a point object, which generates point measurements. Two common approaches to estimating target states from noisy measurements are multiple hypothesis tracking (MHT) [18] and the joint probabilistic data association filter (JPDAF) [1]. Both these methods typically assume that one target produces at most one measurement, and that any measurement is produced by at most one target. Fragment and group measurements would violate this assumption. Khan *et al.* [9] use particle filtering to extend the solution to the case of multiple measurements per target (splits) and multiple targets per measurement (merges). Their method is quite computationally expensive. Genovesio and Olivo-Marin [7] propose a

different algorithm that addresses the same problem. However, both these methods assume a fixed number of targets. This assumption simplifies the problem significantly, since the ambiguity mentioned in the introduction disappears.

While a number of generic object detection and tracking methods based on background-subtraction have been proposed, most of them use ad-hoc solutions for grouping foreground pixels and associating them with objects. The most common technique used to address the *fragmentation* problem—where a single object is split into multiple foreground blobs—is to use dilation-erosion operations to cluster blobs that are spatially proximate [22, 10]. This leads to loss in spatial resolution (see Figure 3), and may fail when there are objects of varying sizes in the same scene (*e.g.*, trucks and pedestrians). Some authors have proposed tracking clusters of pixels using spatial Gaussians, and merging and splitting these clusters when they interact [16, 17]. While these methods are well suited for tracking a varying number of objects of different types, they do not label clusters as fragments, objects or groups, or do not perform association of object IDs before and after group interactions. Other methods [15, 19, 23] use a threshold on the maximum distance between connected components (and perhaps also a threshold on difference in velocity) to group foreground blobs into *super-clusters*. Each super-cluster is associated with a single object. Use of a distance-based threshold leads to loss in resolution, and is not very effective in scenes with significant projective distortion or objects with grossly different sizes. An interesting solution to the fragmentation problem has been provided by Cohen and Medioni [4], who use temporal coherence of blobs to associate them with objects. They however do not handle groups of objects. Our framework can be thought of as a generalization of their approach to handle groups produced by occluding/interacting objects. We refer the reader to the article by Gabriel *et al.* [5] for a review of other methods for tracking multiple occluding objects.

There has been some work on tracking groups of objects [2, 3]. Gennari and Hager [6] have proposed a group-tracking algorithm where all measurements are assumed to come from groups. Thus, objects and fragments are also considered as groups. We use an algorithm similar in spirit to theirs for our first level of tracking (target-set tracking: Section 4.1). However, the focus of our work is our inference graph labeling algorithm (Section 4.3), which allows us to distinguish among objects, fragments and groups, and stitch together target-set tracks to obtain object tracks.

2. Outline of our approach

As many of the concepts used in our work are not standard in the tracking literature, we first provide a high level description of our approach in this section. Our discussion is divided into two parts: an intuitive explanation of our problem formulation, followed by an outline of our solution with the help of an example. The full details of these two parts of our work will be given in Sections 3 and 4 respectively.

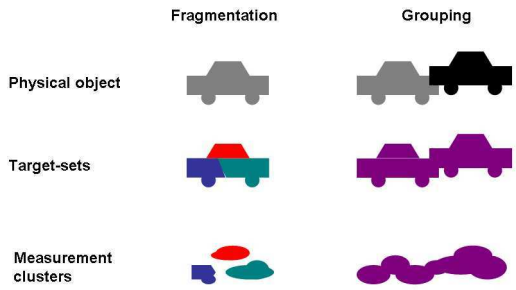


Figure 4. An illustration of the concepts of target-sets and measurement clusters. The true physical objects are in grey and black. The measurement clusters (connected components, in this case) are indicated in different colors. The corresponding target-sets have the same colors. On the left, the object is fragmented into 3 target-sets. On the right, two objects are grouped into 1 target-set.

2.1. The Target-Set Concept

Most tracking methods assume a model of the world where each physical object is represented by a corresponding single target. The targets are treated as hidden variables, but they generate measurements that can be observed. The task of the tracking system is to infer the states of the targets from the observed measurements. If the number of targets is unknown, then additional assumptions are typically made: each target generates at most one measurement, and each measurement is generated by at most one target.

This standard framework is not suitable for our problem, where the number of objects is *unknown* and only generic object models are available. To solve our problem, we model the world as consisting of *sets of elementary targets*. An elementary target corresponds to an indivisible part of a physical object. For instance, each atom of an object can be thought of as an elementary target. Of course, it is not feasible to track a single elementary target. However, sets of elementary targets can be tracked (provided they are large enough to be observed). We use the term *target-set* for a set of elementary targets that are indistinguishable from each other.

We group together foreground pixel measurements (using a method similar to extracting connected-components) into measurement-clusters (or *blobs*). Our observation model is then defined by the following constraint: each *target-set* can produce at most one measurement-cluster. Using this model, we track target-sets with a simple dy-

namical model, till these sets merge or split. Note that by tracking target-sets, we avoid having to explicitly represent individual elementary targets in our framework.

Under ideal imaging conditions, each object being tracked would produce exactly one measurement cluster, and could thus be represented by one target-set. In practice, however, a single object might be split into multiple foreground connected-components. Each such component would then be interpreted as having being generated a distinct target-set (corresponding to a distinct physical part of the object). Such target-sets are called fragments. On the other hand, a single connected-component might be produced by two objects that are interacting with and occluding one another. Such a component would have to be explained by a single target-set that contains elementary targets representing two different objects.

A target-set is thus a generalization of an object, since it can correspond to a part of an object, a whole object, or even multiple interacting objects (see Figure 4). This notion is especially useful in our setting, where we do not know *a priori* whether the measurement-clusters we detect are parts of objects, whole objects, or groups of objects. Even without this knowledge, we are able to track target-sets, till they merge or split to form other target-sets. We then apply spatio-temporal coherence constraints to stitch together the target-set tracks and label them as fragments, objects or groups.

2.2. Summary of our Algorithm

We present here a high-level summary of our approach with the help of an example (Figure 5), to highlight the significant steps. A complete formal description of our solution is presented in Section 4.

We first track foreground pixel clusters (connected components) till merges or splits are detected (Figure 5a). We call this target-set tracking (Section 4.1). We create a directed graph (Section 4.2) representing relationships between tracks of target-sets whenever merges and splits occur (Figure 5b). In general, parent vertices in the graph correspond to merged tracks, while their children contain split tracks.

We define objects (in Section 3.3) as maximal target-sets satisfying certain spatio-temporal coherence constraints. We provide a bottom-up graph-labeling algorithm (Section 4.3) that uses this object definition to stitch target-sets tracks and label them as objects or groups (Figure 5c). Finally, crossover association of object identities across group interactions is carried out (Section 4.4) to obtain the entire track for each object (Figure 5d).

3. Framework Description

3.1. Problem Definition

The input to our tracking system is a set of measurements, $\mathbf{Z}_{1:T} = \{z_{i,t} | 1 \leq i \leq m_t, 1 \leq t \leq T\}$, where m_t is the number of measurements at time t. In this paper, the measurements are the foreground pixels obtained by background subtraction. We use an improved version [12] of the Stauffer-Grimson background-subtraction algorithm [22] that takes into account spatio-temporal dependencies in the foreground regions.

The output of our system is the set of tracks of objects $\mathbf{Y} = \{Y^i_{t_{i1}:t_{i2}} | 1 \leq i \leq N\}$, where $Y^i_{t_{i1}:t_{i2}}$ is the track of the i^{th} object existing from time t_{i1} to t_{i2}. The total number of objects, N, is not known *a priori*. Note that two objects will share a track when they interact with each other to form a group.

The precise definitions of objects and groups will be given in Section 3.3. Before we provide these, we first formally introduce the concept of target-sets.

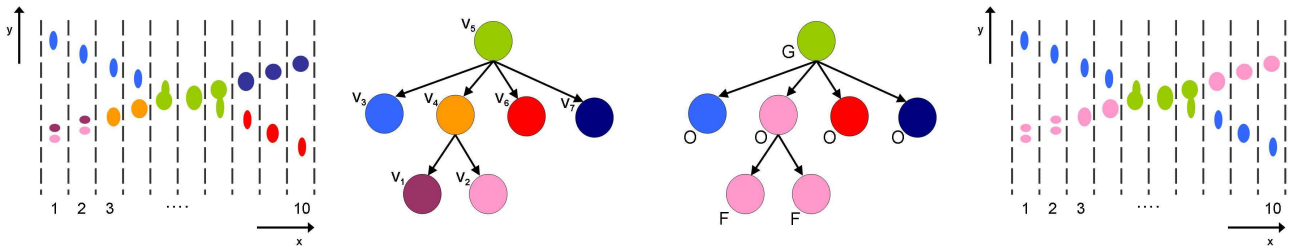
3.2. Target-Sets

We define the target-set $A_{i,t}$ as the i^{th} collection of elementary targets at time t, such that all these elementary targets are indistinguishable from each other given the measurements. For distinct i and j, $A_{i,t} \cap A_{j,t} = \emptyset$. A target-set could be an *object*, a part of an object (*fragment*), or the union of several objects (*group*).

We partition the foreground pixel measurements $z_{i,t}$ at time t into K_t clusters, $C_{k,t}$. Two foreground pixels belong to the same cluster if they fall into a common $s \times s$ window. These clusters can be thought of as a generalization of 8-connected components ($s = 3$) to *windowed* connected components (arbitrary s). We consider measurements within a cluster to be indistinguishable, in order to avoid the complexity of associating individual pixels with target-sets. Typically, the number of measurement-clusters is two or three orders of magnitude smaller than the number of measurements.

Our observation model—for generating measurements from target-sets—involves an important constraint: each target-set $A_{i,t}$ can generate at most one measurement cluster $C_{k,t}$ (see Figure 6a). This is consistent with our definition of target-sets, since we consider measurements within a cluster to be indistinguishable. Further details of the model will be given when we discuss our target-set tracking algorithm in Section 4.1.

For each target set $A_{i,t}$, we keep a state vector $x_{i,t} = (u_{i,t}, v_{i,t}, \Lambda_{i,t}, N_{i,t})$. Here, $u_{i,t}$ is the position and $v_{i,t}$ is the velocity of the target centroid. The (spatial) covariance matrix of the elementary targets in $A_{i,t}$ is denoted by $\Lambda_{i,t}$, and $N_{i,t}$ is the expected number of foreground pixels projected from $A_{i,t}$. We also store a target-set label $\ell_{i,t}$ (*ob-*



(a) (b) (c) (d)

Figure 5. An example illustrating our tracking framework. Image (a) shows measurement clusters detected in 10 frames. To show multiple frames in one image, dashed vertical lines indicate regions of the scene where measurements occur in each frame. There are two objects, moving from left to right while crossing each other and occasionally fragmenting. Image (b) shows the inference graph constructed from merge/split events detected in (a). Colors in (a) and (b) represent target-set tracking results. Stitching of tracks and use of the generic object definition allows labeling tracks as fragments (F), objects (O) or groups (G) (image (c)). Colors in (c) show stitched tracks. After associating objects across groups, image (d) shows our final result, indicated by the corresponding colors from image (c).

ject, *fragment* or *group*) and a set of identifiers $\eta_{i,t}$. We use $Q_{i,t} = \{x_{i,t}, A_{i,t}, \ell_{i,t}, \eta_{i,t}\}$ to represent the set of parameters associated with this target-set.

We allow merging and splitting of target-sets, in order to accurately represent the processes of object fragmentation and group interaction. This makes target-set tracking different from the standard target tracking framework. In particular, the correspondence of target-sets between times $t-1$ and t is a bit complicated. We consider five possible cases:

(1) $A_{j,t-1}$ and $A_{i,t}$ are the same set of targets

(2) $A_{i,t-1}$ disappears at time t, no target in $A_{i,t-1}$ will ever reappear

(3) several target-sets at time $t-1$ merge into one target-set at time t, $A_{i,t} = \bigcup_j A_{j,t-1}$

(4) one target-set at time $t-1$ splits into several target-sets at time t, $A_{i,t-1} = \bigcup_j A_{j,t}$

(5) $A_{i,t}$ is a set of new targets, none of which exist before t

Note that we forbid the case where a target-set is involved simultaneously in a merge and a split at time t. The reason for this constraint will be explained in Section 4.2.

3.3. Objects, Fragments, and Groups

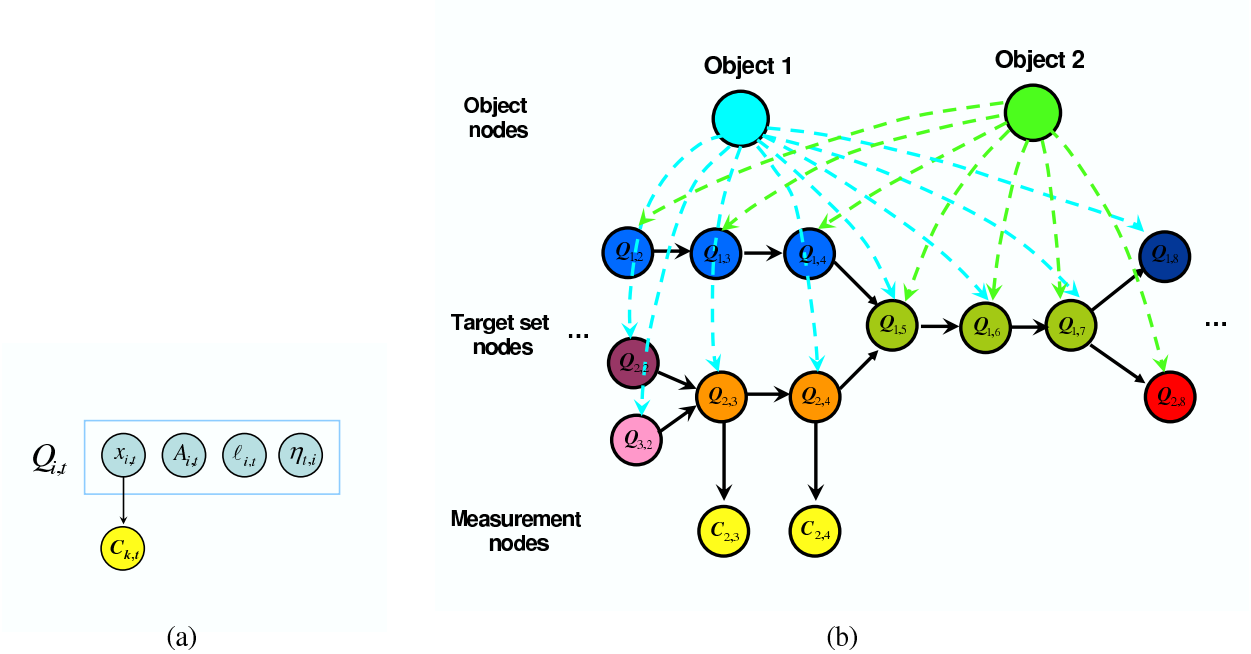
We are now ready to precisely define the concepts of *object*, *fragment*, and *group*. These are 3 types of target-sets. An **object** is defined as a maximal set of elementary targets that are *close in space* and exhibit *coherent motion*. Being close in space means that, for each pair of elementary targets belonging to the object, there exists some time t such that these targets belong to the same target-set $A_{i,t}$. Coherent motion means that all the elementary targets belonging to the object move with approximately equal average velocity over extended periods of time. In other words, the maximum difference in (average) velocity between any pair of elementary targets from the same object is less than a fixed threshold. Essentially, this definition provides a model for (the 2D projection of) a physically-connected 3D object, while allowing for the fact that at every time t, some of its parts may not be detected. The relative configuration of the parts should be stable over time, in the sense that the distance moved by these parts relative to each other is much smaller than the overall distance moved by (the centroid of) the object.

The other two types of target-sets can be defined with reference to *objects*. A **fragment** is a subset of an object. A **group** is a target-set consisting of elementary targets from two or more different objects. Loosely speaking, *groups* form when multiple objects interact with and occlude each other. In our framework, every target set $A_{i,t}$ can be assigned one of these three labels.

3.4. Generative Model

The state of the world can be parametrized as $\mathbf{Q}_{1:T} = \{Q_{i,t} | 1 \leq i \leq n_t, 1 \leq t \leq T\}$, where $Q_{i,t} = \{x_{i,t}, A_{i,t}, \ell_{i,t}, \eta_{i,t}\}$ describes the i^{th} target-set at time t (Figure 6a), and n_t is the number of target-sets at time t. $x_{i,t}$ is the state vector for $A_{i,t}$ (Section 3.2). $\ell_{i,t}$ labels $A_{i,t}$ as *object*, *fragment* or *group*. $\eta_{i,t}$ is the set of object/group identifiers (IDs) associated with $A_{i,t}$. If $A_{i,t}$ is of type *fragment* or *object*, it is associated with only one object ID. If $A_{i,t}$ is a *group*, $\eta_{i,t}$ contains one group ID and multiple object IDs.

The reader should note that the state description in the above paragraph (in terms of $\mathbf{Q}_{1:T}$) is the dual of the description of the output in Section 3.1 (in terms of $\mathbf{Y}$). The former ($\mathbf{Q}_{1:T}$) can be converted to the latter ($\mathbf{Y}$) by searching for all target-sets $A_{j,t}$ that have a common ID i (that is, $\eta_{j,t} = i, \forall j$), and assigning the tracks of all these target-sets



(a) (b)

Figure 6. A graphical model for our tracking framework. (a) For a cluster of foreground pixels $C_{k,t}$ at a particular frame t, there is a target-set $A_{i,t}$ with corresponding parameters $Q_{i,t} = \{x_{i,t}, A_{i,t}, \ell_{i,t}, \eta_{i,t}\}$. (b) Bayes net model, using frames 2 to 8 of the example data in Figure 5(a). There are three types of nodes in the network: object nodes, target-set nodes, and measurement nodes. (Though we do not show them all, there is one measurement node for each target-set node). Colors used for target-set nodes are the same as those in Figure 5. See text for details.

to the i^{th} object Y^i.

Using a few frames from Figure 5(a) as an example, we provide a generative model for our tracking framework in Figure 6b. There are three types of nodes in the model: object nodes, target-set nodes, and measurement nodes. The solid arrows between target-set nodes determine the merge/split events. Solid arrows from target-sets to measurements indicate correspondence. A dashed arrow from an object node to a target-set node means that the target-set either belongs to or contains the object (depending on what type of target-set it is). The edges between an object and its target-set descendants ensure that the latter have identical IDs and exhibit spatial connectedness and coherent motion. If a target-set node has two different objects as parents, it should be labeled as *group*.

In reality, the number of nodes and edges in the model is variable, depending on how many objects are present, and how many fragments and groups are produced by these objects. Thus, inferring the structure of the model is part of the problem. For our model, inferring the number of object nodes and dashed arrows is harder than inferring the solid arrows. Khan *et al.* [9] tried to discover the optimal structure of their multi-object tracking model using Markov Chain Monte Carlo (MCMC) methods, under the assumption that the object number is known and fixed. Even in that simpler case, their solution is computationally expensive. If the object number is varying and unknown, the possible hypotheses are much more complicated. Instead of performing inference directly on this generative model, we propose another model which is a good approximation to this model. In this simpler model, inference can be done in four sequential steps. We discuss this four-step algorithm in the next section.

4. Algorithm

We describe our tracking algorithm in four parts. The first step (Section 4.1) is to track target-sets till they merge or split. At this stage, we do not care whether the tracked target-sets are *fragments*, *objects* or *groups*. The second step (Section 4.2) is to organize the tracked target-sets into vertices of a directed, acyclic graph called the inference graph. The merge/split events detected in the first step are used to add edges in this graph from each merged target-set to target-sets just before the merge and just after the split (as illustrated in Figure 5a,b). Thus, target-sets that are *groups* will be parents (ancestors) of those that are *objects*. Similarly, *fragments* will be children (descendants) of *objects*.

The purpose of building the inference graph is to be able to stitch together target-set tracks that belong to the same object. In the third step (Section 4.3), we do this stitch-

ing, and also label the target-sets, using our inference-graph labeling algorithm. The basic idea is to process the graph bottom-up, stitching tracks together, till the set of stitched tracks violate the 'coherent motion' constraint in the *object* definition. This allows us to detect the *object* target-sets, and hence the *fragment* and *group* target-sets (which are stored at descendants and ancestors of the *object* vertices, respectively). The fourth step (Section 4.4) completes the tracking process by associating *object* tracks across *group* interactions.

4.1. Target-Set Tracking

We develop a target-set tracking algorithm similar to the group-tracking algorithms in Blackman [2] and Gennari and Hager [6]. However, while these algorithms track groups of objects, our target-sets can be objects, fragments, or groups. Besides tracks of target-sets, we also detect merge and split events during target-set tracking.

There are two steps in target-set tracking: measurement association and state estimation. The state of a target-set $A_{i,t}$ is $x_{i,t} = (u_{i,t}, v_{i,t}, \Lambda_{i,t}, N_{i,t})$. The position $u_{i,t}$ and velocity $v_{i,t}$ of the target centroid are modeled as a Gaussian distribution $\mathcal{N}([\bar{u}_{i,t}, \bar{v}_{i,t}], \Sigma^c_{i,t})$ at time t. Spatial locations of individual targets in $A_{i,t}$ are modeled as a Gaussian distribution $\mathcal{N}(u_{i,t}, \Lambda_{i,t})$. Given the association between measurements and target-sets, states are updated by Kalman filtering.

To associate measurements with target-set $A_{i,t-1}$, a search region $SR_{i,t}$ at time t is defined as

$$SR_{i,t} = \{a | (a - \bar{u}^p_{i,t})^T {\Lambda_{i,t-1}}^{-1} (a - \bar{u}^p_{i,t}) < \gamma\}$$

where $\bar{u}^p_{i,t}$ is the predicted mean centroid position of $A_{i,t-1}$. States of target-sets at time t are then updated as follows:

(1) If only one measurement cluster $C_{k,t}$ lies in $SR_{i,t}$, keep tracking $A_{i,t-1}$. Update its position and velocity by Kalman filtering using the centroid of $C_{k,t}$. Set

$$\Lambda_{i,t} = (1-\alpha)\Lambda_{i,t-1} + \alpha\, Cov(C_{k,t})$$

$$N_{i,t} = (1-\alpha)N_{i,t-1} + \alpha\, |C_{k,t}|,$$

where $Cov(C_{k,t})$ and $|C_{k,t}|$ are the sample covariance matrix and the cardinality of the measurement cluster $C_{k,t}$. α is the update-rate.

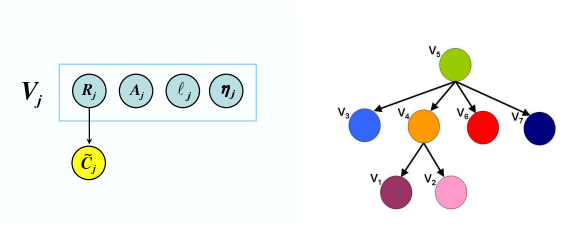
(2) If no measurement cluster lies within $SR_{i,t}$, update the position and velocity of $A_{i,t-1}$ using the dynamic prediction model, and set $\Lambda_{i,t} = \Lambda_{i,t-1}$ and $N_{i,t} = N_{i,t-1}$. If $A_{i,t-1}$ is associated with no measurements for L successive frames, it will be terminated. We thus allow missed detections for a few frames.

(3) If more than one cluster of measurements lies within $SR_{i,t}$, a new target-set is associated with each cluster and initialized. This involves setting the position of each new target-set as the centroid of the corresponding measurement cluster $C_{j,t}$ and the velocity as zero. $\Lambda_{j,t}$ and $N_{j,t}$ are initialized as $Cov(C_{j,t})$ and $|C_{j,t}|$. We stop tracking $A_{i,t-1}$. A split event S is recorded.

(4) If several target-sets $A_{j,t-1}$ share the same cluster of measurements at t, a new target-set $A_{i,t}$ is associated with that cluster and initialized. We stop tracking the sets $A_{j,t-1}$. A merge event M is recorded.

(5) If a cluster of measurements $C_{k,t}$ does not lie within any $SR_{i,t}$, a new target-set is associated with it and initialized.

No target-set is allowed to be simultaneously involved in more than one merge or split event. If that happens, we disassociate the target-sets and clusters of measurements that were involved in the simultaneous merge-split. The simultaneous merge and split events are not recorded. The tracks for the existing target-sets are terminated, and new target-set tracks are initiated and associated with these measurement clusters.



(a) (b)

Figure 7. Structure of the Target-Set Inference graph. (a) The target-set nodes (from the generative model of Figure 6) whose hidden variables $A_{i,t}$ are equivalent are merged into a single vertex V_j of the inference graph. Each vertex $V_j = \{R_j, A_j, \ell_j, \eta_j\}$. A_j is the target-set represented by this vertex. ℓj labels A_j as *object*, *fragment* or *group*. η_j is the set of identifiers associated with A_j. $R_j = \{x_{i,t} | A_{i,t} = A_j\}$ is the track (set of state vectors) associated with the target-set A_j. $\tilde{C}_j = \{C_{i,t} | A_{i,t} = A_j\}$ is the set of measurements generated by the same target-set over time. (b) The target-set inference graph for the data of Figure 5a.

4.2. Target-Set Inference Graph

Given the merge and split events detected by target-set tracking, we build a Target-Set Inference Graph $\mathcal{G}$ representing the relations among different target-sets.

We formally define a merge or split event as a 3-tuple (ε, Ω, t), where ε is the track of a target-set (*i.e.*, the set of state vectors of the target-set over time), Ω is a set of tracks of several target-sets, and t is the time when the merge or split happens. The target-set corresponding to ε is the union of the target-sets corresponding to the tracks in Ω. Merge and split events must satisfy two properties:

PE1: For a merge event $M = (\varepsilon, \Omega, t)$, $\forall \omega \in \Omega$, $end(\omega) = t - 1$ and $start(\varepsilon) = t$; for a split event $S = (\varepsilon, \Omega, t)$, $\forall \omega \in \Omega$, $start(\omega) = t$ and $end(\varepsilon) = t - 1$.

PE2: For any two different merge or split events, $(\varepsilon_1, \Omega_1, t)$ and $(\varepsilon_2, \Omega_2, t)$, $(\{\varepsilon_1\} \cup \Omega_1) \cap (\{\varepsilon_2\} \cup \Omega_2) = \emptyset$.

Here, $start(\varepsilon)$ and $end(\varepsilon)$ are the starting and ending times of track ε. Note that property PE2 means that we avoid simultaneous merge and split of the same target-set.

The inference graph $\mathcal{G}$ is a directed graph, in order to represent the subset relationships between the target-sets, as determined by the merge-split events. As shown in Figure 7, each vertex V_j in $\mathcal{G}$ has a set of hidden variables, $V_j = \{R_j, A_j, \ell_j, \eta_j\}$. A_j is the target-set represented by this vertex. ℓ_j labels A_j as *object*, *fragment*, or *group*. η_j is the set of identifiers associated with A_j. ℓ_j and η_j will be filled later in Section 4.3 and Section 4.4. $R_j = \{x_{i,t} | A_{i,t} = A_j\}$ is the set of state vectors associated with the same target-set A_j over time; we call it the *track* of that target-set. $\tilde{C}_j = \{C_{i,t} | A_{i,t} = A_j\}$ is the set of measurements generated by the same target-set over time. Variables $x_{i,t}$, $A_{i,t}$, and $C_{i,t}$ have been explained in Figure 6. A directed edge from V_i to V_j means that A_j is a subset of A_i.

In order to stitch tracks using a bottom-up algorithm in Sections 4.3 and 4.4, we require $\mathcal{G}$ to have the following properties:

PG1: If two target sets are equivalent, their tracks should be associated with the same vertex.

PG2: One vertex is the descendant of another vertex iff its target-set is the subset of the latter's target-set.

Graph $\mathcal{G}$ is developed in three steps. First, we build a simpler directed graph, $\mathcal{G}_1$, to represent the merge/split events. We modify this graph so that it satisfies property PG1, thus obtaining graph $\mathcal{G}_2$. We then further modify graph $\mathcal{G}_2$ in order to satisfy property PG2, thus obtaining the final inference graph $\mathcal{G}$.

We start by describing the creation of graph $\mathcal{G}_1$. In Section 3.2, we keep the track for each target-set until it disappears, splits, or merges with other target-sets. For each target-set track, we add one vertex V_j to $\mathcal{G}_1$. The corresponding variable R_j is set equal to that track. For each merge or split event (ε, Ω, t), let V_i be the vertex associated with the target-set for track ε. Following the definition of merge/split events and $\mathcal{G}$, we add arrows from V_j to each vertex V_i associated with some track $\omega \in \Omega$. Using properties PE1 and PE2 for merge/split events, $\mathcal{G}_1$ has the following property:

PG3: $\mathcal{G}_1$ is a polytree, *i.e.*, it is acyclic and any two vertices with the same parent have no common descendants.

Note that without property PE2, PG3 is invalid, and a loop might be introduced in our inference graph, making the inference harder. This is why we avoid simultaneous merges and splits during target-set tracking.

The target-sets represented by leaves will not be further decomposed into subsets in $\mathcal{G}_1$, so we call them Target-Set Units (TSUs). The target-set A_i of V_i will be represented in terms of TSUs. For each V_i we keep a temporary variable Δ_i storing a set of TSUs, $\Delta_i = \{U_{i_1}, ..., U_{i_K}\}$, where $A_i = \bigcup_{k=1}^{K} U_{i_k}$. If V_i is a leaf, there is only one TSU in Δ_i representing A_i itself. If V_i has several children, $\{V_{i_1}, ..., V_{i_L}\}$, and all the children are from a single merge or a single split event, we set $\Delta_i = \bigcup_{j=1}^{L} \Delta_{i_j}$. A more complicated case is when a set of tracks A, stored at a set of vertices Γ_A, merges into one track C, and C splits into another set of tracks B, stored at a set of vertices Γ_B (*e.g.*, Figure 8(a,b)). We know that the union of target-sets does not change before and after the merge-split, *i.e.*, $\bigcup_{i \in \Delta_A} U_i = \bigcup_{j \in \Delta_B} U_j = U'$, To represent this equivalence, we create a Target-Set Equivalence Chain (*e.g.*, Figure 8(c)). Each node in the chain is a set of TSUs Δ, such that for two sets Δ_A and Δ_B on the same chain, the union of each set of TSUs represents the same set of targets. We create a new TSU U' for the set of targets represented by the chain, and set $\Delta_C = \{U'\}$. We update Δ_i for each V_i in $\mathcal{G}_1$ in bottom-up fashion, *i.e.*, leaves to roots.

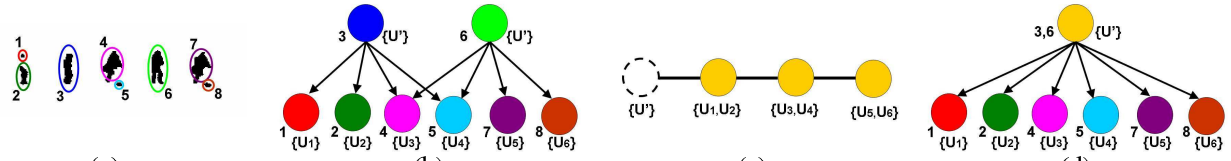
We build graph $\mathcal{G}_2$ from $\mathcal{G}_1$. If two vertices V_i and V_j represent the same set of targets, *i.e.*, they have the same set of TSUs, we replace them with a single new vertex V_k, with $R_k = R_i \cup R_j$, and $A_k = A_i = A_j$. V_k has outgoing edges to all the children of V_i and V_j, and incoming edges from all the parents of V_i and V_j. An example of how to build $\mathcal{G}_1$ and $\mathcal{G}_2$ is shown in Figure 8.

Finally, we build the graph $\mathcal{G}$ from $\mathcal{G}_2$. If V_i is not a descendant of V_j, but A_i is a subset of A_j, we add an edge from V_j (or one of its descendants) to V_i. If two root vertices have a common descendant, we add a virtual vertex, which is not associated with any track, as their common parent. We do this to satisfy PG2. $\mathcal{G}$ becomes a directed acyclic graph (DAG) instead of a polytree.

In Figures 9 and Figure 10, we give two examples to help readers understand the process of building the inference graph.

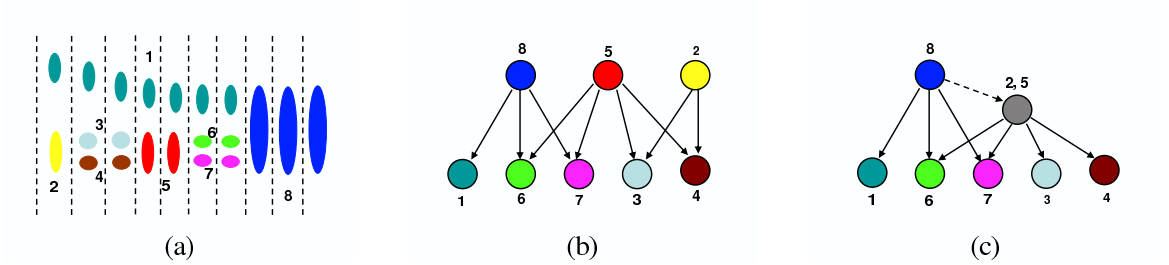
4.3. Inference Graph Labeling Algorithm

So far, we have tracked target-sets and represented the relationships between these tracks using an inference graph $\mathcal{G}$. Each vertex in $\mathcal{G}$ represents a target-set that needs to be labeled as *fragment*, *object* or *group*. Since $\mathcal{G}$ is a directed acyclic graph, child-parent relationships can be defined. By construction, we expect the children W_i of a vertex V to contain tracks generated by the same physical object(s) as the one(s) producing the tracks in V. For instance, if the



(a) (b) (c) (d)

Figure 8. An example showing the steps in building the Target Set Inference Graph, for a case which requires a Target-Set Equivalence Chain. We consider a foreground image sequence showing a person walking. In (a) we show keyframes from the tracks of target-sets. Track 5, for instance, corresponds to the foot of the person, which is tracked separately from the rest of the person's body, before merging with track 4 to form track 6. The graph $\mathcal{G}_1$ for this case is shown in (b). Each vertex in $\mathcal{G}_1$ corresponds to one track in (a). The target-set of each leaf vertex is represented by one TSU $\{U_i\}$. Since tracks 1 and 2 merge into 3 which splits into 4 and 5 again, $U_1 \cup U_2 = U_3 \cup U_4$. Similarly, $U_3 \cup U_4 = U_5 \cup U_6$. We build a Target-Set Equivalence Chain (shown in (c)) to represent the knowledge that $U' = U_1 \cup U_2 = U_3 \cup U_4 = U_5 \cup U_6$. At parent vertices 3 and 6 in (b), we store the new TSU U' which represents the chain. Since the target-sets of vertices 3 and 6 in (b) are represented by the same set of TSUs $\{U'\}$, we replace them with a single vertex in (d) and obtain graph $\mathcal{G}_2$.



(a) (b) (c)

Figure 9. An example of building an inference graph. (a) Target-set tracks (shown in the same single image format as Figure 5a). Based on the merge-split events detected during target-set tracking, an initial inference graph $\mathcal{G}_1$ is built in (b). The two target-sets represented by two separate vertices, 5 and 2, are actually equivalent. Both of them are subsets of the target-set represented by vertex 8. However, there are no arrows from vertex 8 to vertices 5 and 2 in $\mathcal{G}_1$. (c) After performing some operations on the target-set variables A_j (as described in Section 4.2), graph $\mathcal{G}$ is built where vertices 2 and 5 are merged into a single new vertex (marked in gray), and an arrow is added from vertex 8 to the new vertex.

tracks in V come from a single physical object, then the target-sets of all child vertices W_i must be of type *fragment*, and must contain tracks generated by the same physical object.

Algorithm 1. Inference graph labeling algorithm (using functions defined in Algorithms 2–5)

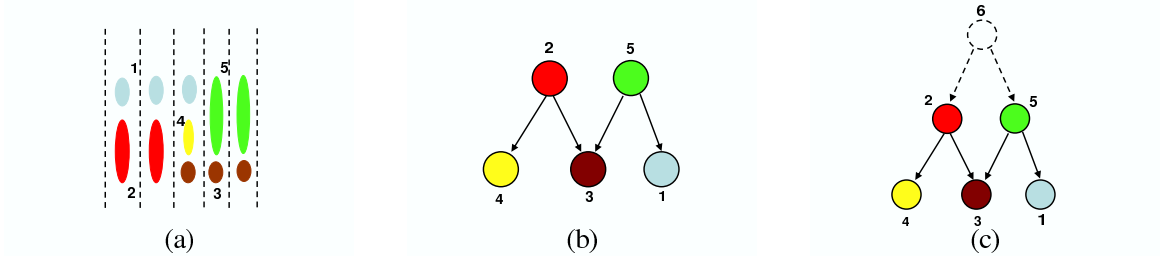
1: **initialize_variables()**
2: **process_bottom_up()**
3: **update_vertex_labels()**
4: **assign_sids_to_tracks()**

In this section, we describe the process for stitching together target-set tracks that belong to the same object, and recovering the labels of the target-sets. We propose a graph inference algorithm that associates tracks in a bottom-up manner—stitching child tracks with parent tracks—till the *coherent motion* constraint defining an *object* is violated by the stitched tracks. In this way, target-sets that should be labeled *object* are identified. Once *objects* have been labeled, *groups* and *fragments* can also be labeled, based on parent-child relationships in the graph. In particular, ancestors of an *object* or a *group* must be *groups*, and descendants of an *object* must be *fragments*. Also, each *fragment* must have an *object* ancestor. Pseudocode for the entire process is provided in Algorithms 1 through 5.

The input to the labeling algorithm consists of the inference graph $\mathcal{G}$. The desired output is three-fold:

1. a *stitched* ID (or sID) ρ assigned to each stitched target-set track in $\mathcal{G}$,
2. a track label $\beta(\rho)$ for each sID ρ, indicating whether the track corresponds to a single physical object ($\beta(\rho) = O$) or a group of objects ($\beta(\rho) = G$)
3. for each sID ρ with $\beta(\rho) = G$, a list of sIDs of all object tracks contained in that group

Once two target-set tracks are stitched together, they will be assigned the same sID. Note that there are two kinds of track labels—*object (O)* and *group (G)*. For instance, in Figure 5d, object tracks are blue and pink, while the group track is green.



(a) (b) (c)

Figure 10. Another example of building an inference graph. (a) Target-set tracks. (b) An initial inference graph $\mathcal{G}_1$ is built based on merge-split events detected in target-set tracking. The target-sets represented by vertices 2 and 5 are not equivalent but have overlap. (c) After the inference among the target-set variables A_j, graph $\mathcal{G}$ is built. A new virtual vertex 6 is added which has arrows pointing to both 2 and 5. Readers can imagine this case as that of a person that starts as two fragments: head (1) and body (2). The body further splits into trunk (4) and feet (3). Finally the three fragments merge into feet (3) and upper-body (5). Since this person never appears as a single blob, no vertex in $\mathcal{G}_1$ could completely represent him. In $\mathcal{G}$, the virtual vertex 6 could represent the person by collecting all the pieces together, although no real track is associated with vertex 6.

In addition to track labels, we define five kinds of vertex labels, for use in intermediate stages of our algorithm: *object* (O_S), *fragment* (F_S), *group* (G_S), *fragment-of-group* (G'_S) and *possible-object* (P_S). The first three vertex labels correspond to the definitions given for types of target-sets in Section 3.3. The *fragment-of-group* (G'_S) vertex label is used for those children W of a *group* vertex V that move coherently with all other children of V. Such child vertices W may consist of *fragments* from multiple physical objects, or may be objects that move together and thus cannot be distinguished. Note that, in either case, the union of target-sets of all child vertices labeled G'_S is equivalent to the target-set of the *group* parent. We need this special vertex label because one step in our algorithm is different for labels G'_S and G_S.

The *possible-object* (P_S) vertex label is given to the children W_i of a vertex V when we find that the tracks of W_i do not exhibit coherent motion (and thus cannot all belong to the same object). Note that we cannot immediately label these children as *objects*, since some of them might actually be *fragments*. This ambiguity is resolved later in the algorithm, when each such vertex is assigned one of the other four vertex labels.

We now discuss the four stages of our algorithm.

4.3.1 Initialization

At each vertex V, vertex-label L_V is initialized to *fragment*.

The *parent-decision* variable $p(W, V)$ of child-vertex W relative to parent-vertex V (which is a binary variable that indicates whether the target-set at W moves coherently with the target-sets stored at the other child-vertices of V) is initialized to zero, indicating coherent motion. This variable is used later in the algorithm to assign final labels to *possible-objects*.

For each vertex $V = (R, A, l, \eta)$, we create and store a copy R' of the track list R. This copy will be updated when tracks broken by merge/split events are stitched together.

Algorithm 2. Pseudocode for the function 'initialize_variables()' called by Algorithm 1

```
1:
2: FUNCTION initialize_variables()
3: for all vertices V = (R, A, l, η) in G do
4:     initialize L_V = F_S
5:     p(V, W_i) = 0 for all parents W_i of V
6:     make a copy R' of V's track list
7: end for
8: ρ_max = 0
```

4.3.2 Bottom-Up Processing

The main body of the algorithm checks each vertex V, to test whether its children move coherently. A vertex can only be checked once its children (if any) have all been checked. This is equivalent to bottom-up processing in a directed acyclic graph.

At each selected vertex V, we proceed as follows. If V is a leaf, we mark it as checked, and move on to another vertex. If V is not a leaf, and some child of V is a *group*, then V is labeled *group* too. In any case, we must test the children of V to see if they belong to the same object. This is done by testing if, for each pair of children, the two sets of child tracks have similar average velocity (*i.e.*, exhibit coherent motion). If any two children W_i and W_j have an average velocity difference exceeding a threshold T_{vel}, they are labeled as *possible objects* (unless they are already *groups* themselves), and V is labeled a *group* vertex[1]. Note that spatial connectedness of tracks of W_i and

[1] If tracks of two children do not overlap for a sufficiently long period of time (say, 5 frames), T_{vel} is set to infinity. For instance, if the tracks

W_j is already guaranteed, since these children share a common parent. Thus, our object definition (Section 3.3) is being used to distinguish objects from groups. W_i and W_j are not immediately updated to *objects*, since they might each be *fragments* of larger objects. However, since they cannot belong to the same object, their parent must be a *group*. We set $p(W_i, V) = 1$ and $p(W_j, V) = 1$, implying that V is a *group* vertex, and that W_i, W_j possibly contain tracks of entire *objects*.

After checking each vertex V, the tracks stored in the updated track lists at all its children are appended to the updated list R' stored at V. In this way, complete track information is available at V for future checks involving its parents.

Algorithm 3. Pseudocode for the function 'process_bottom_up()' called by Algorithm 1

```
FUNCTION process_bottom_up()
while there exists at least one unchecked vertex do
    find an unchecked vertex V = (R, A, l, η) which has no unchecked children
    if V is not a leaf then
        if for some child W of V, L_W = G_S then
            set L_V = G_S
        end if
        for all i, j such that W_i and W_j are children of V do
            d_V(i, j) =average difference in velocity between tracks of W_i and W_j
            if ||d_V(i, j)|| > T_vel then
                if L_{W_i} ≠ G_S, set L_{W_i} = P_S
                if L_{W_j} ≠ G_S, set L_{W_j} = P_S
                set p(W_i, V) = 1 and p(W_j, V) = 1
                set L_V = G_S
            end if
        end for
        add all tracks stored at all children to the new list of tracks R' at V
    end if
    mark vertex V as checked
end while
```

4.3.3 Updating Vertex-labels

After checking all vertices, any root vertex V_0 still labeled *fragment* corresponds to a set of tracks from a single object that does not interact with any other object. Hence, the label of V_0 is upgraded to *object*. Next, if any *group* vertex V has a *possible-object* child W such that $p(W, V) = 0$, the tracks of W represent coherent motion with other children of V, and are therefore *fragment-of-group* tracks. However, any vertex V labeled *possible-object* with $p(V, W) = 1$ for all parent vertices W is upgraded to *object*, since its tracks are not coherent with any other set of tracks.

do not overlap at all, they are automatically considered to be coherent with respect to each other.

Algorithm 4. Pseudocode for the function update_vertex_labels() called by Algorithm 1

```
FUNCTION update_vertex_labels()
for all root vertices V such that L_V = F_S do
    L_V = O_S
end for
for all vertices V such that L_V = G_S do
    for all children W of V such that L_W = P_S and p(W, V) = 0 do
        set L_W = G'_S
    end for
end for
for all vertices V such that L_V = O_S do
    if p(V, W) = 1 for all parents W of V then
        set L_V = O_S
    end if
end for
```

4.3.4 Assignment of sIDs and Track-labels

sIDs and track labels are assigned to all tracks stored at vertices whose vertex-labels are of type *object*, *group* or *fragment-of-group*. Note that because of the previous 2 steps, all tracks must be listed at at least one such vertex. For vertices labeled *object* and *fragment-of-group*, tracks listed in the updated track lists are marked as *object* ($\beta = O$) and *group* ($\beta = G$) tracks respectively. Vertices labeled *group* are treated slightly differently from the other two: only the initial tracks listed at these vertices are marked as *group* tracks (*i.e.*, $\beta = G$). This is because, at a *group* vertex, the updated track-list R' includes *object* tracks. Further tracks of *fragment-of-group* vertices are assigned the same ID as tracks at their *group* parent. Finally, for each sID corresponding to a *group* track, we create a list of sIDs of all objects belonging to that *group* track.

4.4. Obtaining Object Tracks after Crossover Association

For objects that are not involved in any group interactions, the stitched tracks obtained in the previous section constitute our final result. However, for objects that do interact, the stitched object tracks ($\beta = O$) will either begin (in a split) from or end (in a merge) with a group track ($\beta = G$). The whole track of each such object is now obtained by associating stitched tracks before and after the

Algorithm 5. Pseudocode for the function assign_sids_to_tracks() called by Algorithm 1

```
1:
2:  FUNCTION assign_sids_to_tracks()
3:  for all vertices V = (R, A, l, η) such that L_V ∈
    {O_S, G_S} do
4:     ρ_max = ρ_max + 1
5:     if L_V = O_S then
6:        for all tracks r' ∈ R', set ρ(r') = ρ_max
7:        set β(r') = O
8:     else {L_V = G_S}
9:        for all tracks r ∈ R, set ρ(r) = ρ_max
10:       set β(r) = G
11:       for all children W = (R_W, A_W, l_W, η_W) of V
          such that L_W = G'_S do
12:          for all tracks r' ∈ R'_W, set ρ(r') = ρ_max
13:          set β(r) = G
14:       end for
15:    end if
16: end for
```

group interaction. This is done using information about the merge/split events detected in Section 4.1.

First, we estimate the object state from its stitched tracks. To this end, we find all the target-set tracks $A_{j,t}$ that have been assigned the same sID i. The state Y_t^i of the object i at time t is estimated by combining the states $x_{j,t}$ of the target-set tracks $A_{j,t}$, weighted by their sizes $N_{j,t}$. Next, we go through the merge/split events and find all the cases where n objects merge into one group and and then split into n objects. We need to do crossover association between tracks ε_i before the merge and tracks ω_j after the split. In more complicated cases, some objects first merge into a group, which further merges with other objects or groups into a bigger group. Then the big group splits into smaller groups and finally into objects. In such cases, our inference graph can help us correctly identify the two sets of n objects between which to do crossover association. One example can be find in our results section.

Having identified two sets of n objects that need to be matched, many matching techniques can be used. We use a simple technique, wherein, for each object ε_i, we predict its position after the split (using Kalman filtering), and compare it with the position of ω_j. We pick the best one-to-one matching between the two sets. Once ε_i and ω_j are matched, both their sIDs are re-mapped to a new object ID. Our simple technique works well when multiple objects interact by occluding each other, while still maintaining independent trajectories. More complicated, long-term interaction cases could be handled by using other features, such as color, shape and texture for matching; we have not considered these. The focus of our work is on being able to identify the sets of objects that interact to form the group. Once this is done, any suitable matching technique can be used.

Currently, we cannot do crossover association when, according to the results of our labeling algorithm, n objects merge into a group and split into m objects ($m \neq n$). If we assume that the number of objects should not change during interaction, at least one of the tracks labeled as object by our algorithm should be re-labeled as group. However, such a correction needs more complicated reasoning, perhaps involving the appearance of objects. This is not addressed in the present work. The results section will show one failure case caused by this problem.

Finally, for the periods when an object is merged with other objects into a group, we use the track of the group as the object's track.

5. Experimental Results

We tested our multi-object tracking algorithm on mostly outdoor video sequences captured at resolution 720x480 pixels and 30 fps. The algorithm was tested on more than 30,000 frames (15 minutes) of video. Background-subtraction parameter values were the same as those used by Migdal and Grimson [12]. To obtain measurement clusters from foreground pixels, a window size $s = 11$ was used[2]. Target-set tracking parameters γ and α were set at 4 and 0.8 respectively. The threshold on average velocity difference between tracks, T_{vel} (Section 4.3), was set at 0.5 pixels/frame.

Noise filtering was done in two steps:

1. after background subtraction, removing measurement clusters less than 30 pixels in size, and
2. after target-set tracking, removing target-set tracks that were almost stationary for a long time (more than 90 frames)

In addition to removing stationary clutter, the second condition also allowed us to filter out *ghosts* left behind by objects that were stationary for a while and then moved.

Our algorithm runs relatively fast, even though it is a batch-processing algorithm. For instance, the average running time of our tracking algorithm in MATLAB on a 3.2 GHz PC for 30-second video-clips is around 40 seconds. In Figures 11, 12, 13, 14 and 15, we show representative tracking results. These results illustrate challenging cases which are not easily solved using other methods.

The first example (Figure 11) shows 12 frames sampled from a sequence of inter-person interactions, where there are multiple crossover events. Object IDs are maintained

[2] In general, a large window size produces fewer clusters, and thus speeds up computation, at the cost of lower spatial resolution in distinguishing between objects.

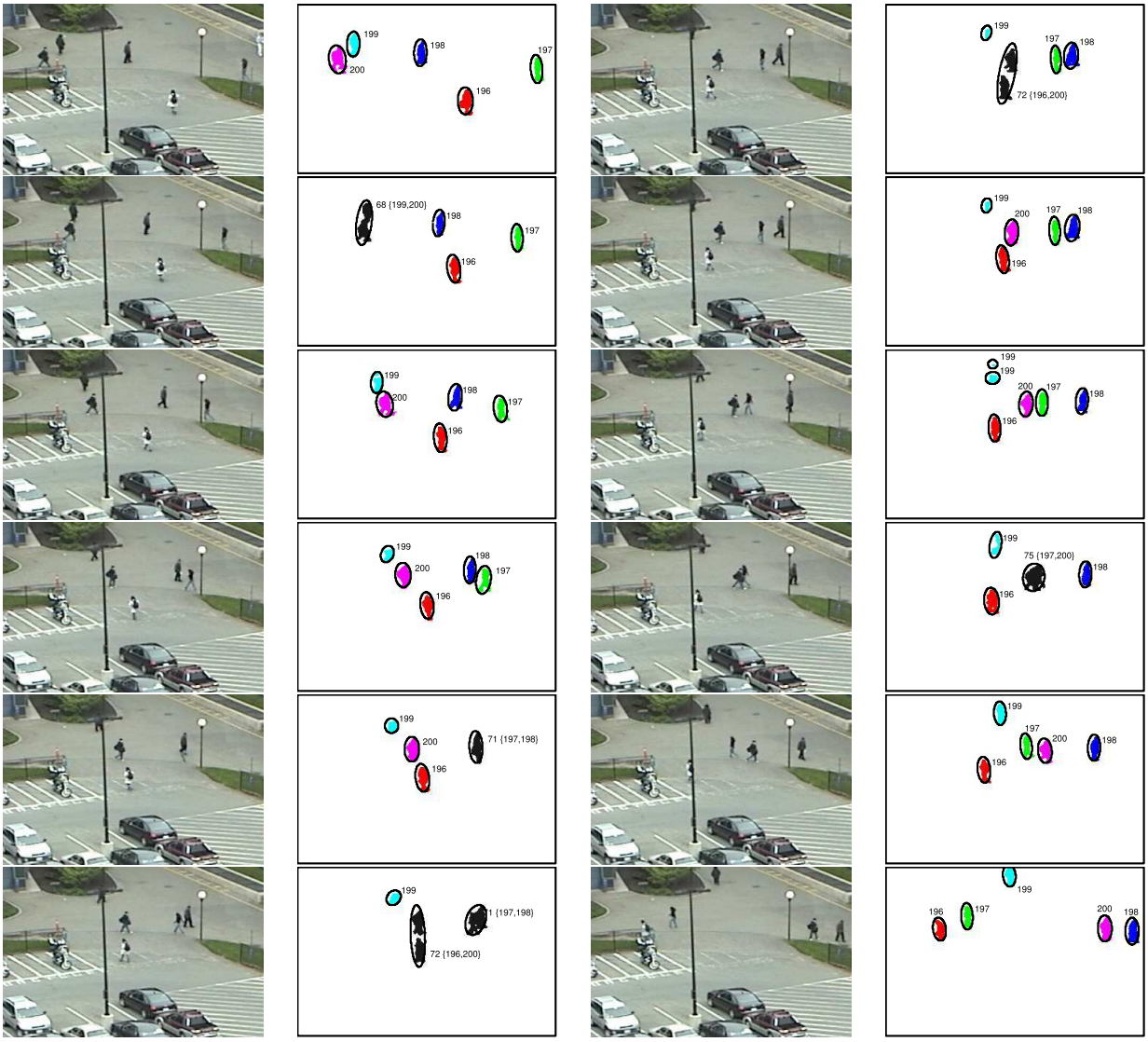


Figure 11. Tracking results on an example involving multiple successive merges and splits. Sampled video frames are shown in columns (a) and (c), in temporal order from top to bottom, and from left to right. The corresponding foreground pixels (after noise filtering) shown next to them in columns (b) and (d). Black ellipses on the foreground images indicate estimated states of tracked target-sets. Colors (and numbers) indicate final IDs assigned to *objects/groups* after labeling the inference graph and performing crossover association. *Groups* are marked in black, and the IDs of the *objects* belonging to a *group* are shown in braces. Note that, in this scene, people sometimes occlude each other significantly while crossing.

across all interactions. Further, the groups correctly keep track of the objects present within them. It is important to recover the complete trajectory of a single object through multiple occlusions, in order to perform activity analysis.

The next example (Figure 12a,b) shows interaction between two different object classes: car and person. Once again, each type of object is tracked successfully through the crossover. The changing view presented by the turning car, which would be difficult to handle with a vehicle-specific appearance model, poses no significant challenge to our system.

We tested our algorithm on an indoor video sequence (Figure 12c,d), after changing the noise filtering threshold to 400 pixels. In spite of almost complete occlusion, our tracker was able to correctly detect the two persons and maintain identities through the crossover interaction. Note that object 50 is often split into 2 fragments with a large gap between them, so simple dilation-erosion operations would

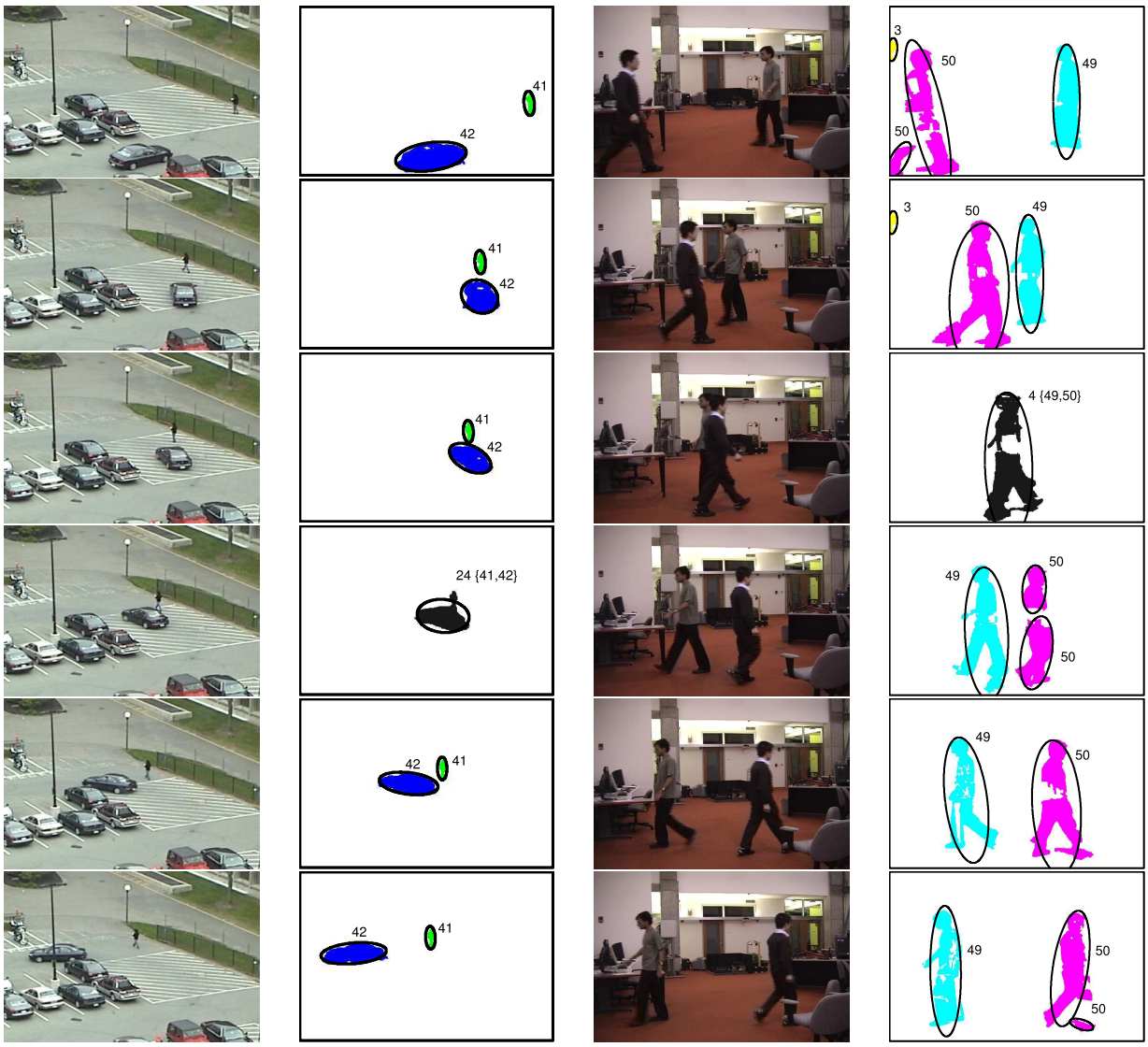


Figure 12. Tracking results for two cases (in the same format as Figure 11). Columns (a) and (b) show a scene with two different object classes, car and person, in an interaction event. Note that the car is rotating, and thus would be hard to track with a vehicle model. Columns (c) and (d) show an indoor sequence with almost complete occlusion and signifi cant fragmentation.

not be able to connect them.

The example in Figure 13a,b shows two persons entering the scene together as a group, so that they produce merged observations from the beginning. After walking together for some time, the persons move apart. We are able to correctly label both the initial group and the objects when they split. Methods that initialize object models with the first few frames of a new track are likely to fail here.

Figure 13c,d shows a case where 3 *objects* interact. Note that *object* 170 is actually a pair of persons, who walk together and are thus indistinguishable. First, *objects* 170 and 171 merge to form a *group*. Then, *object* 172 merges with this *group*. Next, *object* 171 separates from the *group*, leaving behind a *group* consisting of *objects* 170 and 172. Finally, the latter *group* splits into the two individual *objects*. Our crossover association algorithm (Section 4.4) is able to correctly maintain object identities throughout this interaction.

The next example (Figure 14) shows a situation where fragmentation is a serious problem. Here, the bus is similar in colour to the background, because of the dark shadow cast by a building. Further, it is much larger in size than the other objects in the scene (car and pedestrian). The background subtraction results show parts of the bus being

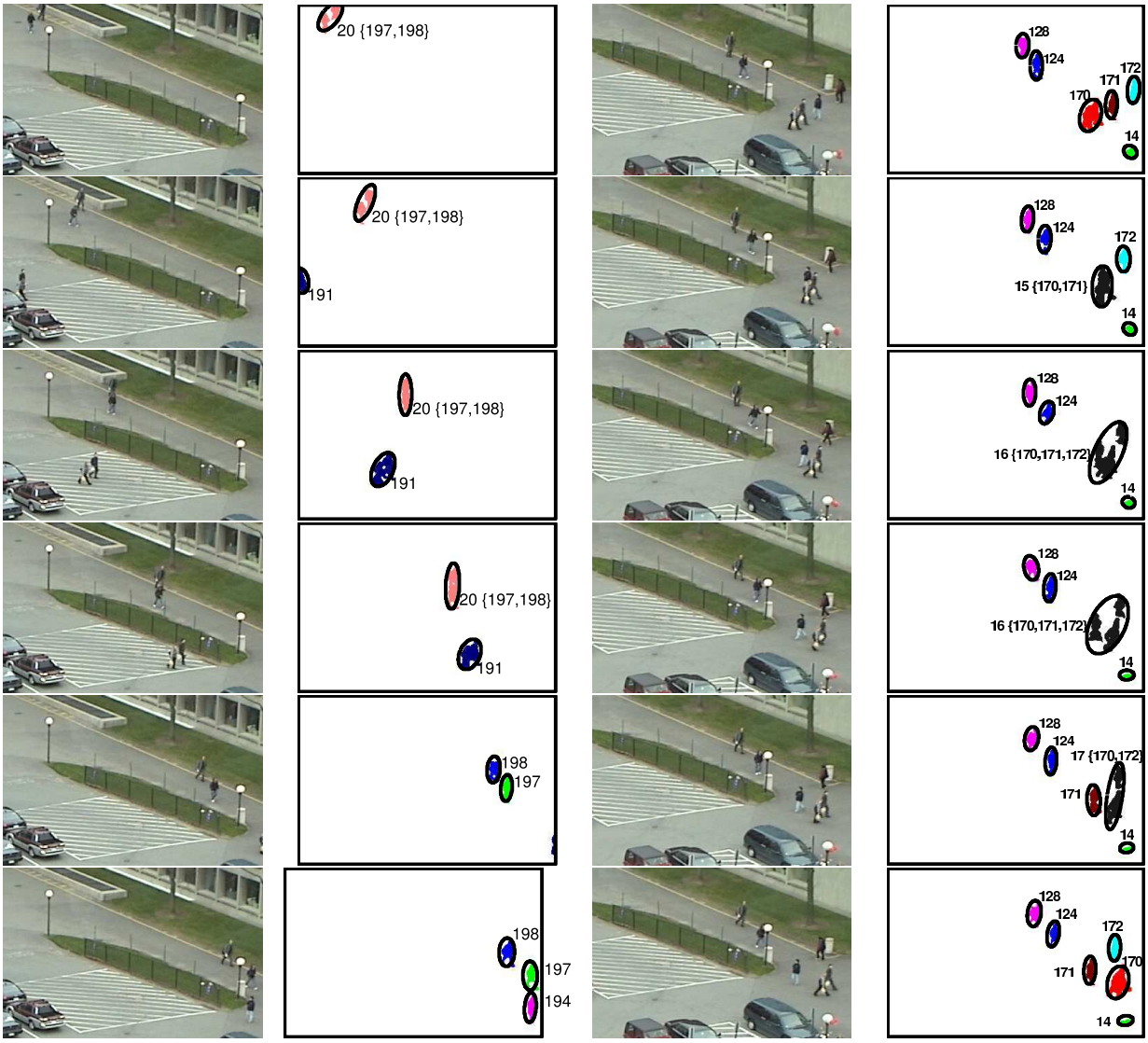


Figure 13. Tracking results for two more cases (in the same format as Figure 11). Columns (a) and (b) show a case where 2 persons enter the scene together, and are detected as a group, but later move apart, and are detected as individual objects. Columns (c) and (d) show a scene with a group of persons crossing a single person.

detected as fragments. Note that there is a large gap between these fragments, which is comparable to the typical gap between pairs of distinct objects. Thus, these fragments cannot be easily grouped using distance-based criteria. Our method, though, correctly associates the fragments with the same object, while still tracking the other objects separately.

Our final example (Figure 15) is more challenging, and is used to illustrate some limitations of our algorithm. The image sequence shows a number of soccer players moving close to each other. Tracking is correct for the first 6 sample frames shown. In the 7^{th} frame, a merge and split involving *object* 90 and *group* 92 occur simultaneously, resulting in incorrect ID assignment to *object* 346 and *group* 101. Another error occurs in the 11^{th} frame, where two persons (the referee and a red-shirted player) are considered to be *fragments* of *object* 352. The reason for this is that, in subsequent frames, these two persons have similar motion, till one of them merges into another *group*. Similar errors were observed in other crowded situations, where merges and splits occur frequently, so that there is not enough trajectory information to distinguish *objects* from *fragments*. Note also that, in this case, *group* 97 is formed when one *object* (461) and a *group* (94) of two objects merge, but that it subsequently splits into two *objects* (351 and 352).

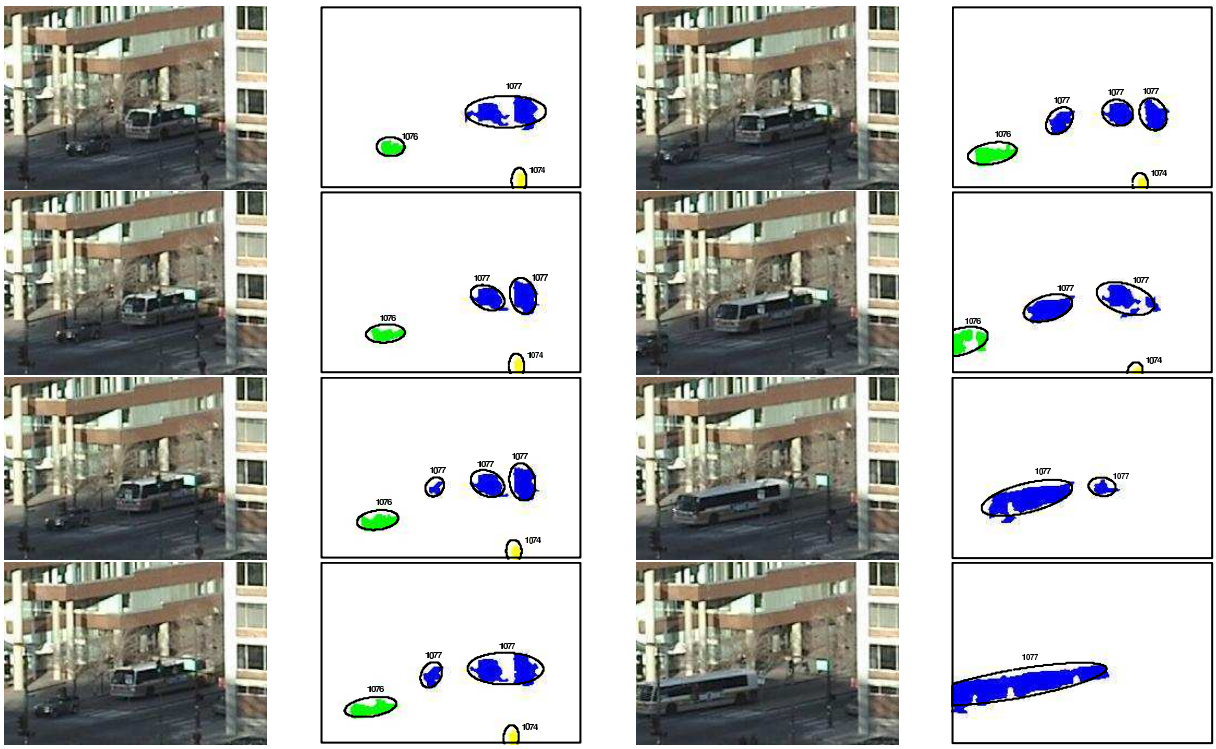


Figure 14. Tracking results for a traffic-intersection scene (in the same format as Figure 11). Note that the foreground image of the bus typically consists of 2-3 fragments (colored blue) that are separated by a large distance. Our algorithm is able to correctly assign all these fragments to a single object.

Crossover association in this case is not handled by our algorithm, since the number of objects is not conserved during the interaction. As a consequence, there are two lists of IDs associated with *group* 97 in the 10^{th} frame.

We evaluated the performance of our algorithm on a 5-minute-long video sequence from the scene shown in Figure 12a. Ground truth was obtained manually by looking through the video. In total, there were 38 moving objects, including cars, pedestrians, balloons tied to a lamp-post, and a dog. From the background subtraction results, it was found that the average gap between fragments was one-quarter of the width of the object. Target-set tracking indicated that, on average, one merge or split event occured every 22 frames.

On 12 occasions, two or more objects crossed in such a way that one of the objects was occluded. Each of these 12 groups was correctly detected. In all but one case, the crossover association (*i.e.*, identity maintenance) was done correctly. The error involved a woman who was carrying a dog before the crossover, but who set the dog down afterwards, so that the dog wandered away some distance. As a result, she and the dog were detected as a single object before the interaction, but as a group of objects after it. Our algorithm currently does not allow associating an object with a group.

There were 3 cases of non-crossover (*i.e.* long-term) interaction. The first involved a car that stopped to pick up two persons, and then drove away. The car was (correctly) labeled as *group* after picking up the persons. The second involved two persons approaching each other, stopping to talk for a while, and then walking back towards where they came from. The third involved the woman and her dog mentioned above. They walked separately for a while, but occasionally the dog came close enough to be occluded by the woman. In the last 2 cases, crossover association did not work. As mentioned earlier, this could be improved by considering object appearance/shape, in addition to position/velocity, but this was not the primary focus of our work.

In general, situations in which our tracking/labeling algorithm may fail include the following:

- If an object is moving before an interaction begins, but is stationary after the interaction ends, the target-set track corresponding to the object after interaction is removed as noise.
- The number of objects in groups may not be con-

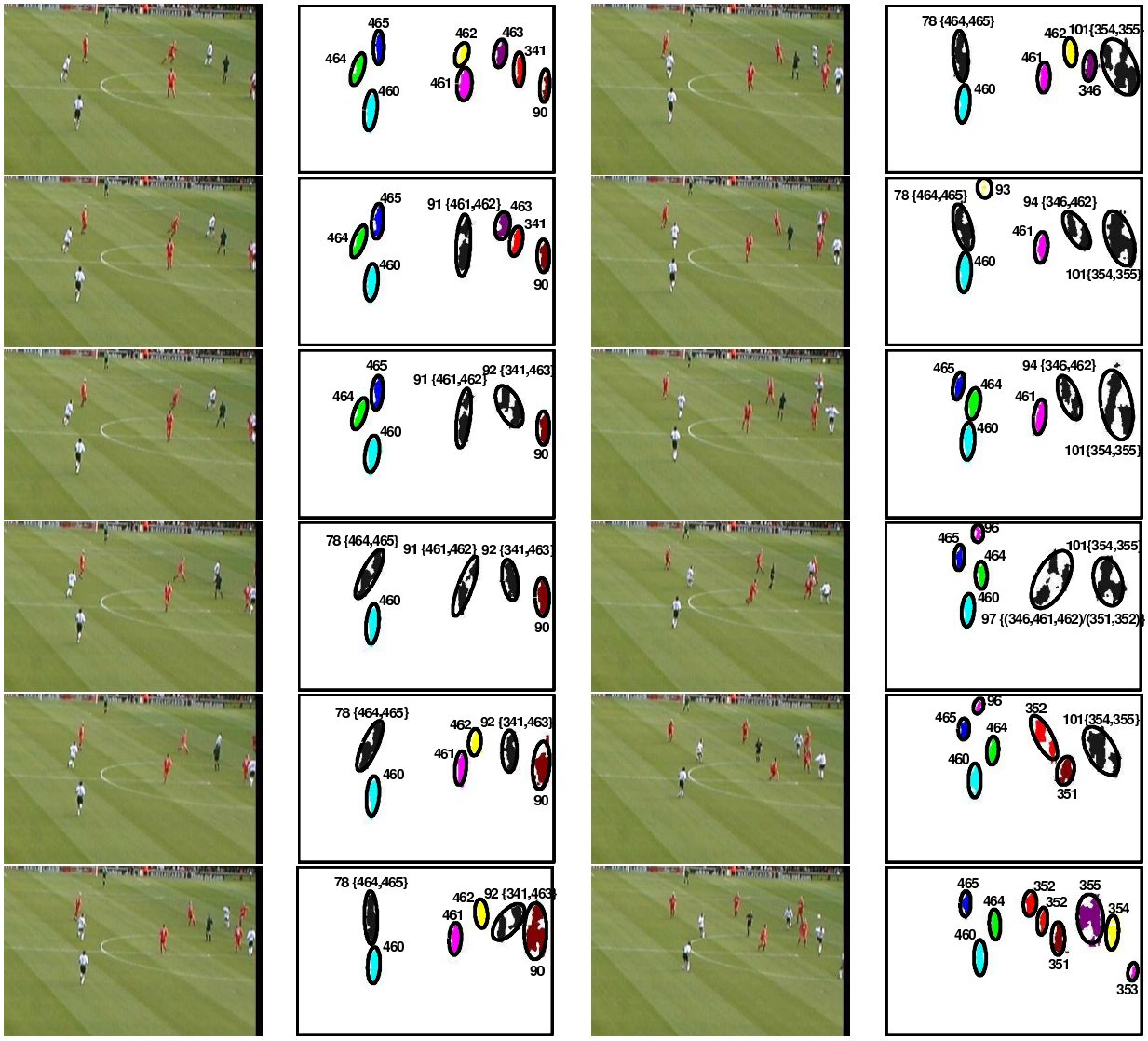


Figure 15. A more challenging data set: soccer data from PETS 2003 (shown in the same format as Figure 11).

served. For instance, if two objects enter the scene as a group, then merge with another object to form a larger group, and finally the large group splits into three individual objects, the initial group will still be labeled as a single object.

- If the same target-set is involved in a merge and a split in the same frame, target-set tracks will be broken (but otherwise correct).

- Reflections (in windows) of moving objects get tracked as independent moving objects.

6. Discussion

We wish to discuss two additional aspects of our work which distinguish it from other current trackers.

6.1. Online versus Batch Algorithms

Our tracking algorithm, as presented here, works in batch mode. However, it can easily be converted to an online algorithm, if necessary. For each new frame, only some target-sets tracks are updated. Only the corresponding relevant parts of the inference graph need to be modified. Track stitching and labeling decisions will then be updated as more evidence arrives. Note that, in either case, it is important to store tracking information from multiple pre-

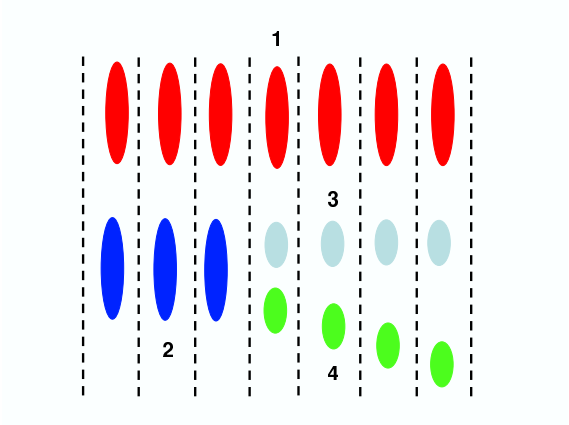


Figure 16. An example to explain uncertainty reasoning in our framework. See text for details.

vious frames, since it is not possible to distinguish objects from fragments and groups using only one or two frames.

6.2. Modeling Uncertainty while Tracking

Many existing trackers are based on a well-defined probabilistic model, and essentially perform (approximate) statistical inference on this model to obtain tracks of objects. The structure of probabilistic dependence in these models is typically chosen so that inference is computationally tractable. For instance, first-order Markov models are a popular choice (in Kalman filters and particle filters). Here, given the current state of the object, its state in the next frame is conditionally independent of its state in previous frames. Such models are suitable for tracking a fixed number of objects in the absence of fragments and groups. Once the number of objects becomes variable, and fragments and groups are allowed, additional multi-frame constraints on object state—namely spatial and temporal coherence—need to be imposed. It would be very computationally expensive to design a probabilistic model to capture this long-term dependence. Instead, we choose to break our model into three parts:

1. a probabilistic model of target-set tracking, for which we employ Kalman filters,
2. a deterministic, constraint-based model of target-set track stitching and labeling, and
3. a search for the best crossover association across group interactions.

As for the first part, more advanced uncertainty reasoning techniques, such as MHT and JPDAF, could be used to improve this framework. However, it is not the focus of this paper. In the second part, many tracking frameworks keep multiple hypotheses, because they want to sequentially integrate evidence from multiple frames to reach the optimal decision. For example, in Figure 16, assuming that we have know in advance that there are three objects in the scene, we cannot decide in the first few frames which of the two tracks (1 or 2) has two objects. Thus two hypotheses have to be temporarily kept. The ambiguity cannot be resolved until we observe that track 3 split into two tracks 3 and 4 later on. If the object number is unknown, as it is in our setting, more hypotheses have to be kept at the beginning. However, in our inference graph, we stitch tracks over multiple frames at each vertex. It turns out that to label a vertex, we only need to check the motion constraint with its siblings and among its child vertices. This allows us to perform bottom-up processing on the graph. Thus, in our framework, we do not need to keep multiple hypotheses over track labels.

In the third part, we search for the best crossover association across interactions, after having identified the groups and the objects they consist of. In this way, we avoid having to keep track of states of individual objects while they are interacting, which can be a difficult task.

In our experience, most of the uncertainty in the model lies in the observation process (since background subtraction is often unreliable). In contrast, a simple dynamical model is sufficient. Further, there are typically enough observations of an object to unambiguously classify it, along with the fragments and groups associated with it.

7. Conclusion

We introduced the concept of target-sets, which we believe are a good representation for tracking multiple objects in the presence of fragments and groups. We also presented an algorithm for tracking a varying number of interacting objects without employing class-specific object models. Interaction events are handled by group tracking, where the members of the group are associated before and after the interaction event. Only a single, uncalibrated camera is used, and objects can be tracked through significant view change. Maintaining identity across multiple interactions makes it possible to consider higher level tasks such as activity recognition and analysis, as well as surveillance tasks involving following a particular object of interest.

In our current implementation, the input is obtained as the set of foreground measurements from a static-camera background subtraction system. In principle, though, any other motion segmentation system could be used to provide the input. For instance, our tracking framework should work just as well for input from a PTZ- or airborne-camera background subtraction system.

References

[1] Y. Bar-Shalom and T. Fortmann. *Tracking and Data Association*. Academic Press, 1988. 2

[2] S. Blackman. *Multiple-Target Tracking with Radar Applications*. Artech House, Dedham, MA, USA, 1986. 3, 7

[3] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, Norwood, MA, USA, 1999. 2, 3

[4] I. Cohen and G. Medioni. Detecting and tracking objects in visual surveillance. In *Proc. CVPR*, 1999. 3

[5] P. F. Gabriel, J. G. Verly, J. H. Piater, and A. Genon. The state of the art in multiple object tracking under occlusion in video sequences. In *Advanced Concepts for Intelligent Vision Systems (ACIVS)*, 2003. 3

[6] G. Gennari and G. D. Hager. Probabilistic data association methods in visual tracking of groups. In *Proc. CVPR*, 2004. 2, 3, 7

[7] A. Genovesio and J.-C. Olivo-Marin. Split and merge data association filter for dense multi-target tracking. In *Proc. ICPR*, 2004. 2

[8] M. Isard and J. MacCormick. BraMBLe: A bayesian Multiple-Blob tracker. In *Proc. ICCV*, 2001. 2

[9] Z. Khan, T. Balch, and F. Dellaert. Multitarget tracking with split and merged measurements. In *Proc. CVPR*, 2005. 2, 6

[10] A. J. Lipton, H. Fujiyoshi, and R. Patil. Moving target classification and tracking from real-time video. In *IEEE Wkshp. Applic. Comp. Vis. (WACV)*, 1999. 3

[11] O. Masoud and N. P. Papanikolopoulos. A novel method for tracking and counting pedestrians in real-time using a single camera. *IEEE Transactions on Vehicular Technology*, 2001. 2

[12] J. Migdal and W. E. L. Grimson. Background subtraction using markov thresholds. In *IEEE Wkshp. Motion Video Comput.*, 2005. 4, 12

[13] A. Mittal and L. Davis. M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene. *Intl. J. Comp. Vis.*, 51(3):189–203, 2003. 2

[14] K. Okuma, A. Taleghani, N. de Freitas, J. J. Little, and D. G. Lowe. A boosted particle filter: Multitarget detection and tracking. In *Proceedings ECCV*, 2004. 2

[15] A. Pece. From cluster tracking to people counting. In *3rd IEEE Intl. Wkshp. Performance Evaluation of Tracking and Surveillance (PETS)*, 2002. 2, 3

[16] A. E. C. Pece. Tracking of non-gaussian clusters in the pets2001 image sequences. In *2nd IEEE Intl. Wkshp. Performance Evaluation of Tracking and Surveillance (PETS)*, 2001. 3

[17] J. H. Piater and J. L. Crowley. Multi-modal tracking of interacting targets using gaussian approximations. In *2nd IEEE Intl. Wkshp. Performance Evaluation of Tracking and Surveillance (PETS)*, 2001. 3

[18] D. Reid. An algorithm for tracking multiple targets. *IEEE Trans. Automatic Control*, 24(6):843–854, 1979. 2

[19] A. Senior. Tracking people with probabilistic appearance models. In *3rd IEEE Intl. Wkshp. Performance Evaluation of Tracking and Surveillance (PETS)*, 2002. 3

[20] N. T. Siebel and S. J. Maybank. Real-time tracking of pedestrians and vehicles. In *2nd IEEE Intl. Wkshp. Performance Evaluation of Tracking and Surveillance (PETS)*, 2001. 2

[21] K. Smith, D. Gatica-Perez, and J. Odobez. Using particles to tracking varying numbers of interacting people. In *Proc. CVPR*, 2005. 2

[22] C. Stauffer and E. Grimson. Learning patterns of activity using real-time tracking. *IEEE Trans. Patt. Anal. Mach. Intell.*, 22(8):747–757, 2000. 1, 2, 3, 4

[23] D. Xie, W. Hu, T. Tan, and J. Peng. A multi-object tracking system for surveillance video analysis. In *Proc. ICPR*, 2004. 3

[24] T. Yu and Y. Wu. Decentralized multiple target tracking using netted collaborative autonomous trackers. In *Proc. CVPR*, 2005. 2

[25] T. Zhao and R. Nevatia. Tracking multiple humans in crowded environment. In *Proc. CVPR*, 2004. 2