Multi-class object tracking algorithm that handles fragmentation and grouping

Biswajit Bose Xiaogang Wang Eric Grimson
MIT Computer Science and Artificial Intelligence Laboratory
{cielbleu|xgwang|welg}@csail.mit.edu

Abstract

We propose a framework for detecting and tracking multiple interacting objects, while explicitly handling the dual problems of fragmentation (an object may be broken into several blobs) and grouping (multiple objects may appear as a single blob). We use foreground blobs obtained by background subtraction from a stationary camera as measurements. The main challenge is to associate blob measurements with objects, given the fragment-object-group ambiguity when the number of objects is variable and unknown, and object-class-specific models are not available. We first track foreground blobs till they merge or split. We then build an inference graph representing merge-split relations between the tracked blobs. Using this graph and a generic object model based on spatial connectedness and coherent motion, we label the tracked blobs 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 during interactions. Experimental results on multiple video sequences are shown.

1. Introduction

In an ideal setting, a visual monitoring system would track each physically distinct moving object in its field of view – each track would be temporally complete (no gaps over the time period during which the object is in view) and spatially complete (each object would always have a corresponding single, connected silhouette in image space that completely covers only those pixels reflecting the object's position). In practice, this is not always possible, especially in unconstrained, arbitrarily illuminated, far-field settings involving many objects from many different classes. Several issues make this problem difficult: partial occlusion (*e.g.,* trees, fences) and accidental alignment (*e.g.,* portions of a moving object are accidentally very similar to objects in the background resulting in undetected moving pixels) can fragment a silhouette into temporally or spatially separated elements; spatial proximity of objects can cause detected silhouettes to merge temporarily into a single silhouette. As a result, ideal tracks are often fragmented into components[1], or are merged with other tracks into common components. Our goal is to create a framework in which these fragmented and merged track segments are unified to create distinct, complete tracks for each object.

There are several methods for attacking this problem. Most are only appropriate to settings with a single class of object, with limited interactions between objects, or where objects are sufficiently large to support strong appearance models. For example, model-based multi-object tracking methods detect objects from a single class of interest (*e.g.,* tracking humans using person-models [14]). Unfortunately many natural far-field settings do not fit these restrictions. In environments with many object-classes of interest, such as urban scenes with pedestrians, cars, motorbikes, trucks, bicycles and pets, it may not be feasible to use class-specific appearance and dynamics models. Building class-specific models requires sufficient labeled examples from each class or designing all these models by hand. It is especially difficult to make viewpoint/scale-invariant models. Further, for applications such as surveillance, the interesting objects may be ones that have not been seen before.

An alternative is to perform motion-based detection [12] of objects from a stationary camera. These can then be tracked with a *generic* (as opposed to class-specific) object model. However, the results deteriorate when a single object is observed as multiple foreground blobs (fragmentation), or when objects pass near each other so that their observations are merged into a single blob (grouping). While some approaches handle grouping, the problem of fragmentation is either ignored, or avoided by clustering nearby foreground blobs, which leads to loss of spatial resolution.

If the number of objects is varying and unknown, an ambiguity arises when using generic models to track objects that may fragment or group. This **fragment-object-group ambiguity** is illustrated by the foreground blobs in Figure 1(c,d,e). Here, two interacting objects initially merge into a single foreground blob. A frame later, the objects

[1] In experiments on a variety of unconstrained videos, we found fragmentation to occur once every 6 frames, on average.

1-4244-1180-7/07/$25.00 ©2007 IEEE

separate, but one of the objects further splits into two blobs. Finally, whole object appear. This scenario is hard to distinguish from cases where each blob is a whole object, since we simultaneously need to estimate the number of objects and the association of foreground blobs with objects.

This ambiguity makes solving the fragmentation and grouping problems together much harder than solving either independently. Existing generic-object tracking systems often avoid dealing with fragmentation [13], incorrectly associate a single object with measurements from a group of objects, or track fragments of an object as independent objects [12]. Further, many trackers do not maintain identities of object tracks before and after interactions [4, 10].

In this paper, we consider the use of generic object models for tracking multiple interacting objects, while explicitly modeling the process of fragmentation and grouping. Our goal is to process motion sequences to obtain single, connected tracks for each distinct object, even if objects interact or overlap, or if objects pass behind partially occluding barriers, or if accidental alignment of a moving object with the background yields visually indistinct regions. We detect and track objects and groups of objects under the following constraints: (a) a varying and unknown number of multiple classes of objects, (b) object measurements (*i.e.* background subtraction blobs) may be fragmented or grouped, (c) a single, stationary, uncalibrated camera, and (d) arbitrary 3D poses of objects. The motivation for our work is to track a diverse collection of objects in an unrestricted, *open-world* environment, without scene-specific training.

Our approach is to track foreground pixel clusters (*blobs*), until they merge or split. Each tracked blob (called a *target-set*) needs to be labeled as a fragment of an object, a whole object, or a group of objects, since in any given frame it could be any of these cases. To infer the correct label, we need to integrate information from nearby frames. To do this, we define a generic object model based on *spatial locality* and *coherent motion* of constituent parts. Using this definition, and a graph-based representation of tracks and their merges and splits, we infer the labels of target-sets, and therefore the tracks of objects and groups[2]. Finally, we associate object tracks before and after group interactions. Thus, we maintain tracks of all moving objects with correct identities through multiple interaction events.

This paper makes two key contributions. The first is our generic object definition, which can be used to disambiguate objects from fragments and groups, by using only detected foreground blobs from multiple frames. The second is an algorithm that tracks the blobs, and uses our object definition to associate and label them.

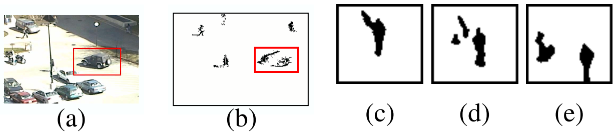


Figure 1. (a,b) Car silhouette has big fragments (comparable in size to people), with large gap in between. (c,d,e) (Zoomed-in) foreground frames, showing the fragment-object-group ambiguity.

1.1. Related work

In the radar tracking community, where class-specific models are often not available [1], objects are often treated as point targets, generating point measurements. Khan *et al.* [6] extend this method to the case of multiple measurements per target (fragments) and multiple targets per measurement (groups). Their method is computationally expensive, even though it assumes a fixed number of targets. Genovesio and Olivo-Marin [5] propose an alternate algorithm, that also assumes a fixed number of targets.

Most generic-object tracking methods based on background-subtraction use ad-hoc solutions for associating foreground blobs with objects. The most common solution to the *fragmentation* problem is to use a distance-based criterion (such as dilation-erosion operations) to cluster blobs or tracks that are near one other [10, 11, 12, 13]. This leads to loss in resolution (see Figure 1a,b), and is not effective in scenes where objects have grossly different sizes (*e.g.* trucks and pedestrians). In a densely populated scene, distance-based clustering of blobs may lead to all the objects being merged into a single group, rendering tracking meaningless. Another possibility is to allow merges/splits while tracking clusters of pixels. These methods either do not distinguish objects from fragments/groups, or do not associate object IDs before and after group interactions. Cohen and Medioni [2] address the fragmentation problem by requiring temporal coherence of blobs. They do not handle groups of objects. Gabriel *et al.* [3] review other multi-object tracking methods.

There has been some work on tracking groups of objects [1]. Gennari and Hager [4] propose a group-tracking algorithm where objects and fragments are not distinguished from groups. We use an algorithm similar in spirit for our first-level blob-tracking (Sec. 3.1). However, our focus is on our labeling algorithm (Sec. 3.3), which allows us to label blob-tracks as objects, fragments and groups. The problem of associating objects before and after group interactions has also been tackled using Bayesian networks [7, 9, 13]. These methods do not handle the case of fragmented objects.

The paper is organized as follows. Section 2 defines the concept of target-sets, and uses it to define our generic object model. It also includes a summary of our tracking/labeling algorithm, whose details are provided in Section 3. The next section contains experimental results.

[2] By tracking groups as distinct entities whenever they occur, we avoid performing occlusion reasoning.

2. Framework Description

The input to our 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 there are m_t measurements at time t. The measurements are foreground pixels obtained by background subtraction [8] (an improved version of [12]) for each video-frame. The output is a 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 number of objects, N, is not known *a priori*. An object track may consist of multiple fragment tracks. Two objects share a track when they interact to form a group.

2.1. The Target-Set Concept

In most tracking methods, each physical object is represented by a corresponding target. This approach is not suitable for handling fragmentation. Instead, we model an object as a *set of elementary targets*. An *elementary target* corresponds to an indivisible part of a physical object. Of course, we can only track *sets* of elementary targets large enough to be observed. So we define the **target-set** $A_{i,t}$ as the i^{th} set of elementary targets at time t that are all indistinguishable from each other given the observed measurements. For $i \neq j$, $A_{i,t} \cap A_{j,t} = \emptyset$. A target-set could be an *object*, a part of an object (*fragment*), or the union of several objects (*group*). The intuition is that the observable events (*image blobs*) may be composed of elements from one or more actual physical objects, which over time could shift from one observed blob to another. We want to track the set of elementary targets that belong to a single object as they move from one observable set to another, to create a unique track for each physical object. Hence we need to label each observable blob in a manner that supports that analysis.

We spatially partition the measurements $z_{i,t}$ at time t into K_t clusters, $C_{k,t}$. Two foreground pixels belong to the same cluster (or *blob*) if they fall into a common $s \times s$ window. These blobs generalize 8-connected components ($s = 3$) to *windowed* connected components. We consider measurements within a blob to be indistinguishable. Note that indistinguishable does not imply the pixels have identical intensities, rather that pixels in a blob are currently part of the same connected moving component. In principle, appearance information such as color could be used to distinguish elements of a blob, however, our experience with far-field settings indicates that color information is often not sufficient to uniquely distinguish subparts of blobs.

To track target-sets, we need data association of blobs with target-sets. In our observation model, each *target-set* $A_{i,t}$ can produce at most one blob $C_{k,t}$. Using this model, we track target-sets[3], till these sets merge or split. From frame $t-1$ to t, target-sets may propagate ($A_{j,t-1} = A_{i,t}$), appear, disappear, merge ($A_{i,t} = \bigcup_j A_{j,t-1}$) or split ($A_{i,t-1} = \bigcup_j A_{j,t}$). Merging and splitting of target-sets allows us to represent the processes of object fragmentation and group interaction.

2.2. Our Generic Object Model

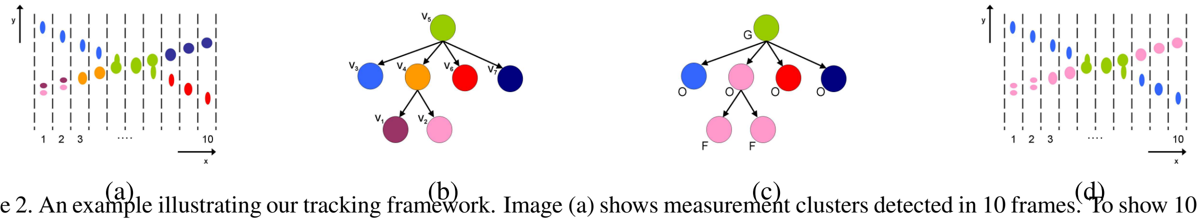
We define 3 target-set labels: *object*, *fragment*, and *group*. An **object** is a maximal set of elementary targets that exhibit *spatial connectedness* and *coherent motion*. *Spatial connectedness* means that for each pair of elementary targets belonging to the object, there exists some time t such that these targets belong to the same target-set $A_{i,t}$, *i.e.,* they are part of the same observed image blob. *Coherent motion* means that all the elementary targets move with similar average velocity over extended periods of time. In particular, we model the (average) difference in speeds (in both horizontal and vertical directions) between any pair of elementary targets from the same object as independently Gaussian-distributed with zero mean and small variance σ_0^2. In contrast, the difference in speeds between 2 targets from different objects is modeled as independent Gaussians with zero mean and large variance σ_1^2. Essentially, this definition provides a model for (the 2D projection of) a physically-connected 3D object, while allowing that, at every time t, some of its parts may not be detected. The relative configuration of the parts should be stable over time.

The other two types of target-sets are 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. Every target-set $A_{i,t}$ can be assigned one of these three labels. We do not know *a priori* whether the tracked target-sets are *fragments*, *objects*, or *groups*. In Section 3, we use the generic object definition given above to design an algorithm that distinguishes between *fragments*, *objects* and *groups*, and obtains complete *object* tracks by stitching together relevant parts of *fragment* and *group* tracks with existing *object* tracks.

2.3. Summary of our Approach

Our algorithm has many details, so we first present a high-level summary using an example (Fig. 2). We first track target-sets (Sec. 3.1) using the foreground blobs till merges and splits are detected (Fig. 2a). At this stage, we do not know whether the tracked target-sets are *fragments*, *objects* or *groups*. The second step (Sec. 3.2) organizes the tracked target-sets into vertices of a directed, acyclic graph called the inference graph. The graph is built so that *spatial connectedness* (*i.e.*, the first part of our object definition) is guaranteed between any node and all of its descendants. 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 before the merge and after the split (see Fig. 2b). Thus, target-sets that are *groups* will be parents (ancestors)

[3] By tracking target-sets, we avoid having to explicitly represent individual elementary targets.



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

Figure 2. An example illustrating our tracking framework. Image (a) shows measurement clusters detected in 10 frames. To show 10 frames in 1 image, dashed vertical lines indicate regions of the scene where measurements occur in each frame. There are 2 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 object model 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).

of those that are *objects*; *fragments* will be children (descendants) of *objects*. Note that these labels are still unknown.

In the third step (Fig. 2c), we use the inference graph to stitch together target-set tracks that belong to the same object, and also label the target-sets, using our inference-graph labeling algorithm (Sec. 3.3). The labeling of target-sets is done using the second part of our object definition: we process the graph bottom-up, stitching tracks together, till the set of stitched tracks violate the *coherent motion* constraint. We thus detect *object* target-sets, and hence *fragment* and *group* target-sets (which are stored at descendants and ancestors of the *object* vertices, respectively). Finally, we associate object identities across group interactions (Sec. 3.4) to obtain the entire track for each object (Fig. 2d).

3. Algorithm

3.1. Target-Set Tracking

We use a target-set tracking algorithm similar to the group-tracking algorithm in Gennari and Hager [4]. We also detect merge and split events during target-set tracking.

A target-set $A_{i,t}$ has a state vector $x_{i,t} = (u_{i,t}, v_{i,t}, \Lambda_{i,t})$, where $u_{i,t}$ and $v_{i,t}$ are the centroid position and velocity. Spatial locations of elementary targets in $A_{i,t}$ are modeled as a Gaussian $\mathcal{N}(u_{i,t}, \Lambda_{i,t})$. To do data association, for each target-set $A_{i,t-1}$, we define a search region $SR_{i,t}$ at time t. If multiple blobs lie within $SR_{i,t}$, a separate target-set $A_{k,t}$ is initialized and associated with each blob. We define $A_{i,t-1} = \bigcup_k A_{k,t}$, stop tracking $A_{i,t-1}$, and record a split event. If search regions of several target-sets $A_{j,t-1}$ share a single blob at t, a new target-set $A_{k,t}$ is initialized and associated with that blob. We define $A_{k,t} = \bigcup_j A_{j,t-1}$, stop tracking the sets $A_{j,t-1}$, and record a merge event. See (Fig. 2a) for example merges and splits. If a target-set merges and splits simultaneously, a virtual frame is inserted in between so that the merge happens first and then the split. For a one-to-one correspondence, $A_{i,t-1} = A_{k,t}$. After data association, states of target-sets are updated by Kalman filtering.

3.2. Target-Set Inference Graph

Given the tracks, and the merge and split events detected above, we build a directed Target-Set Inference Graph $\mathcal{G}$ representing the spatial relations among target-sets. Each vertex V_j in $\mathcal{G}$ corresponds to a target-set A_j[4]. The set $R_j = \{x_{i,t} | A_{i,t} = A_j\}$ of all state vectors associated with the same target-set A_j over time is stored in V_j.

$\mathcal{G}$ is structured so that application of our object definition is easy. It will guarantee *spatial connectedness* between targets stored at any vertex and targets stored at all descendants of that vertex. For this, we require $\mathcal{G}$ to satisfy property **P0**: a directed edge from V_i to V_j exists if the merge/split events indicate that A_j is a subset of A_i.

The *coherent motion* constraint of our object definition involves calculation of target-set track velocities. This requires stitching multiple short target-set tracks that are spatially connected to get complete tracks, from which the relevant velocities can be obtained and the constraint tested. To stitch tracks, we use a bottom-up algorithm (Secs. 3.3 and 3.4), which requires $\mathcal{G}$ to satisfy 2 properties: (**P1**) if two target sets are equivalent, their tracks should be associated with the same vertex; (**P2**) vertex V_j is the descendant of vertex V_i iff A_j is a subset of A_i.

Graph $\mathcal{G}$ is developed in 3 steps. First, we build a simpler graph, $\mathcal{G}_1$, to satisfy property P0. For each target-set track (Sec. 3.1), we add a vertex V_j to $\mathcal{G}_1$. R_j is set equal to that track. When several target-set tracks ω_i merge into or split from a target-set track ε, we add directed edges from the vertex for ε to the vertices for each ω_i. Since we have already modified simultaneous merge-splits, there are no loops, and $\mathcal{G}_1$ is a polytree. See (Fig. 2b) for an example.

To satisfy property (P1), we modify $\mathcal{G}_1$, obtaining $\mathcal{G}_2$, by checking whether the target-sets at different vertices are equivalent. The target-sets U_i at the leaves of $\mathcal{G}_1$ cannot be further decomposed into subsets, so we call them Target-Set Units (TSUs). We use these small sets U_i to infer the equivalence of larger target-sets. The target-set A_i at each

[4] We do not use a time-index t here, since we have defined equality of target-sets over time during target-set tracking

V_i will be represented in terms of TSUs. For each V_i, we store a temporary set of TSUs, $Q_i = \{U_{i_1}, ..., U_{i_K}\}$, where $A_i = \bigcup_{k=1}^{K} U_{i_k}$. If V_i has several children, $\{V_{i_1}, ..., V_{i_L}\}$, from a single merge or split event, we set $Q_i = \bigcup_{j=1}^{L} Q_{i_j}$. When a set of tracks A merges into one track C, and C splits into another set of tracks B, we know that the union of target-sets does not change before and after the merge-split, *i.e.*, $\bigcup_{i \in Q_A} U_i = \bigcup_{j \in Q_B} U_j = U'$. We create a Target-Set Equivalence Chain (TSEC) to represent such equivalences. If target-sets at two vertices V_i and V_j are equivalent, *i.e.*, the unions of their TSUs are equivalent (as found from the TSEC), we replace them with a single new vertex V_k, with $R_k = R_i \cup R_j$, and $A_k = A_i = A_j$.

Finally, we build graph $\mathcal{G}$ from $\mathcal{G}_2$, to satisfy property (P2). 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 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. $\mathcal{G}$ becomes a directed acyclic graph (DAG) instead of a polytree. Fig. 3 shows an example of how to build an inference graph.

3.3. Inference Graph Labeling Algorithm

Each vertex in graph $\mathcal{G}$ represents a target-set that can be labeled as *fragment*, *object* or *group*. In this section, we stitch together target-set tracks that belong to the same object, and assign labels to target-sets. To do so, we enforce the *coherent motion* constraint: we design a bottom-up graph algorithm that stitches child tracks with parent tracks till this constraint is violated by the stitched tracks.

The coherent motion constraint implies that any 2 target-set tracks from the same object have a true average velocity difference vector[5], Δ_t, that is zero-mean Gaussian distributed with diagonal covariance (and small variance σ_0^2 along both directions). Further, the velocity difference between 2 target-set tracks from different objects is zero-mean Gaussian with large variance σ_1^2. An additional source of uncertainty is observation noise: we model the set of observed velocity difference vectors $\mathbf{\Delta_o}$ over N frames as Gaussian distributed about the true velocity difference Δ_t with variance τ^2. Let $c(W_i, W_j)$ be an indicator function that equals 1 if target-sets at vertices W_i and W_j move coherently, and 0 otherwise. We can now determine the probability $p(c(W_i, W_j) = 1|\mathbf{\Delta_o})$ of target-sets at child vertices W_i and W_j (of a parent V) satisfing the coherent motion constraint, given the observed velocity differences $\mathbf{\Delta_o}$:

$$p(c(W_i, W_j) = 1|\mathbf{\Delta_o}) \tag{1}$$

$$\propto p(\mathbf{\Delta_o}|c(W_i, W_j) = 1)p(c(W_i, W_j) = 1) \tag{2}$$

$$= 0.5 \int \prod_{k=1}^{N} p(\Delta_o^k|\Delta_t)p(\Delta_t|c(W_i, W_j) = 1)d(\Delta_t) \tag{3}$$

Equation 3 has a non-informative prior (0.5) over coherent motion of siblings. We similarly calculate $p(c(W_i, W_j) = 0|\mathbf{\Delta_o})$, and normalize the probabilities.

These probabilities will be used in our algorithm to perform a hypothesis test to tell whether the parent vertex V is a group, and whether pairs of children are objects. *Fragments* can then be labeled, based on parent-child relationships in the graph, since descendants of an *object* must be *fragments*. See Fig. 2c for an example labeling.

The input to the algorithm is the inference graph $\mathcal{G}$. The output is: (a) a *stitched* ID (or sID) ρ for each stitched target-set track in $\mathcal{G}$, (b) a track label $\beta(\rho)$ for each sID, indicating the track is from a single object ($\beta = O$) or a group of objects ($\beta = G$), and (c) for each group sID, a list of sIDs of all object tracks in that group.

In addition to labeling vertices as *object* (O), *fragment* (F) or *group* (G), we use two temporary vertex labels in intermediate stages of our algorithm: *fragment-of-group* and *possible-object*. The *fragment-of-group* label is used for children W_i of a *group* vertex V that move coherently with all other children of V. Such child vertices may consist of *fragments* from multiple physical objects, or may be objects that move together and thus cannot be distinguished. The *possible-object* label is given to children W_i, W_j of a vertex V when we find that the tracks of W_i, W_j do not exhibit coherent motion with each other (and thus cannot belong to one object). These labels are not immediately updated to *object*, since some of these might actually be *fragment-of-group*. Tracks at vertices with both these labels are later labeled as *fragment*, *object* or *group*.

We now discuss the 4 stages of our labeling algorithm.

Initialization. At each vertex V, vertex-label L_V is initialized to *fragment*. The *parent-decision* variable $d(W, V)$ of child W relative to parent V is initialized to 0, indicating coherent motion[6]. This variable is used later to assign final labels to *possible-objects*. For each vertex V, we create and store a copy R' of the track list R. This copy will be updated when tracks broken by merges/splits are stitched together.

Bottom-Up Processing. The main step checks each vertex, to test whether its children move coherently. A vertex V is only checked once its children have all been checked.

If V is a leaf, we mark it as checked, and move on. If not, then we test whether V is a group. This happens if a child of V is a *group*, or some pair W_i, W_j of its m children $W_1 \ldots W_m$ violates the coherent motion constraint:

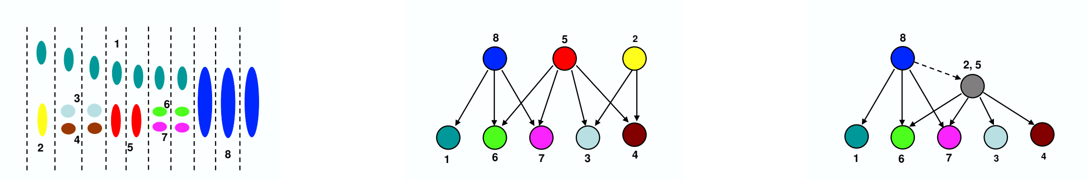
$$p(L_V = G|\mathbf{\Delta_o}) \tag{4}$$

$$= 1 - p(c(W_1, \ldots, W_m) = 1|\mathbf{\Delta_o}) \prod_{k=1}^{m} p(L_{W_k} \neq G)$$

where, $p(c(W_1, \ldots, W_m) = 1|\mathbf{\Delta_o})$ is the probability that

[5] Here *average* implies the mean over all frames in a (stitched) track.

[6] This binary variable indicates whether the target-set at W moves coherently with the target-sets at other children of V.



(a) (b) (c)

Figure 3. An example of building an inference graph. (a) Target-set tracks (shown in same format as Fig. 2a). Using merge-split events detected during target-set tracking, an initial graph $\mathcal{G}_1$ is built in (b). The target-sets represented by vertices 5 and 2 are actually equivalent. Both of them are subsets of the target-set at vertex 8. However, there are no edges from vertex 8 to vertices 5 and 2 in $\mathcal{G}_1$. (c) To build graph $\mathcal{G}$, vertices 2 and 5 are merged into a single new vertex (marked in gray), and an edge is added from vertex 8 to the new vertex.

target-sets at all child vertices move coherently, which we define as $\min(p(c(W_i, W_j) = 1|\mathbf{\Delta_o}))$ over all pairs of children W_i, W_j. If $p(L_V = G|\mathbf{\Delta_o}) > 0.5$, V is labeled *group*. For all pairs of children W_a, W_b such that $p(c(W_a, W_b) = 1) < 0.5$, W_a and W_b are labeled as *possible-objects* (unless they are already *groups* themselves). We set $d(W_a, V) = 1$ and $d(W_b, V) = 1$.

After checking a vertex V, its track list R' is updated by appending the updated track lists of all its children.

Updating Vertex-labels. Any root vertex labeled *fragment* is upgraded to *object*. If any *group* vertex V has a *possible-object* child W such that $d(W, V) = 0$, W is upgraded to *fragment-of-group*. Any vertex W labeled *possible-object* and having $d(W, V_k) = 1$ for all parent vertices V_k is upgraded to *object* (since its tracks are not coherent with any other tracks).

Assignment of sIDs and Track-labels. sIDs and track labels are assigned to all tracks stored in the updated lists at vertices whose vertex-labels are *object*, *group* or *fragment-of-group*. At this stage, all tracks will be listed at at least one such vertex. Tracks stored at *fragment-of-group* vertices are assigned the same sID as their *group* parent. Vertices labeled *group* are treated slightly differently from the other two: only the initial tracks R listed at *group* vertices are marked as *group* tracks (*i.e.*, $\beta = G$). This is because the updated track-list R' at a *group* vertex includes *object* tracks. Finally, for each *group*-track sID, we create a list of sIDs of all objects belonging to that *group*.

3.4. Obtaining Object Tracks

So far, we have labeled all tracked blobs (*i.e., target-sets*) as *fragment*, *object* or *group*. We have also stitched fragment tracks to object tracks, and associated group sIDs to object sIDs. For objects that interact, the whole track is now obtained by matching stitched object tracks before and after the group interaction.

We consider the merge/split events and find all sets of events where n objects merge into one group and then split into n objects. To do crossover matching between tracks ε_i before the merge and tracks ω_j after the split, we predict the position of ε_i after the split (using Kalman filtering) and compare it with the position of ω_j. We find the best one-to-one matching between the two sets of n objects. If ε_i and ω_j are matched, both of their sIDs are re-mapped to a new object ID. Note that, in complicated cases, some objects merge into a group, which further merges with other objects/groups into a bigger group. Then the big group splits into smaller groups and finally into objects. In such cases, our inference graph helps us identify the two sets of objects between which to do crossover association. One example is shown in Fig. 4c,d. Note that other features, such as shape and color, could also be used for matching; we have not investigated these since this is not our primary focus.

4. Experimental Results

We tested our algorithm on more than 60,000 frames of video (720x480 pixels, 30 fps). Background-subtraction parameters were as in [8]. A window size $s = 11$ was used for blob detection, and velocity difference variances (Sec. 3.3) were $\sigma_0 = 3$, $\sigma_1 = 15$ and $\tau = 8$ pixels/frame. Small blobs were removed as noise. After target-set tracking, tracks that were stationary (for more than 30s) were also removed. The average running time of our algorithm in MATLAB on a 3.2 GHz PC for 30s video-clips is around 40s. See supplementary material for tracking videos.

We evaluated the performance of our algorithm on a 15-minute video from the scene in Figure 4c. There were 94 moving objects, including cars, pedestrians, balloons tied to a lamp-post, and a dog; 89 objects were correctly tracked from beginning to end. 762 merges and splits were detected. On 23 occasions, 2 or more objects crossed; these groups were correctly detected. In 1 case, crossover matching was incorrect. Cases of non-crossover interaction included a car picking up 2 persons, and 2 persons approaching each other, stopping and then retracing their steps.

Figures 4, 5, 6 and 7 show results for challenging cases. In these figures, black ellipses indicate estimated states of tracked target-sets. Colors (and numbers) indicate final IDs assigned to *object/group* tracks after crossover association. *Groups* are colored black, with IDs of *objects* in the *group* in

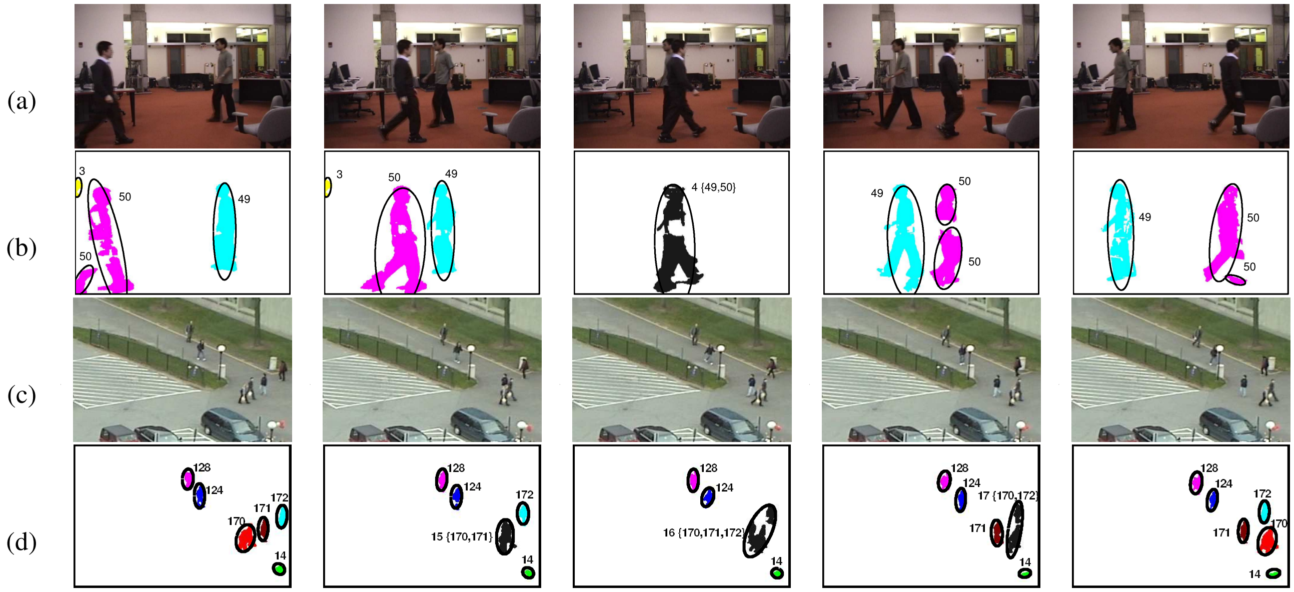


Figure 4. Tracking results for 2 scenes. Sampled video frames are shown in alternate rows, with corresponding foreground pixels (after noise filtering) shown below them. Black ellipses indicate estimated states of tracked target-sets. Colors (and numbers) indicate final IDs assigned to *object/group* tracks after crossover matching. *Groups* are black, with IDs of *objects* in the *group* in braces. See text for details.

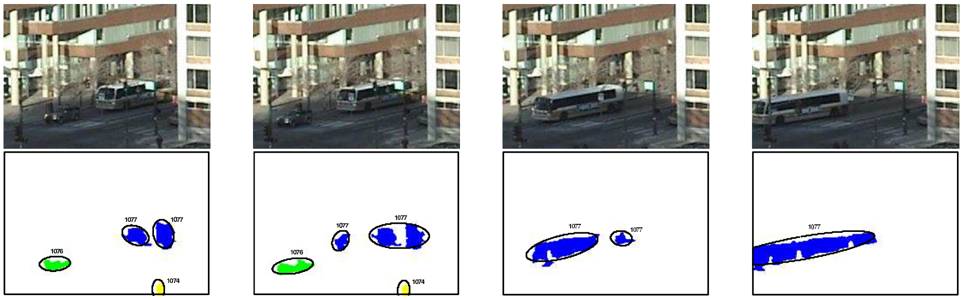


Figure 5. A traffic-intersection scene (in the same format as Figure 4). See text for details.

braces. In Fig. 4a,b, despite almost complete occlusion, our tracker correctly detected 2 persons and maintained identities through the crossover. Note that object 50 is often split into 2 fragments with a large gap between them. Fig. 4c,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. *Objects* 170 and 171 merge to form a *group*, which merges with *object* 172. Eventually, the *group* splits into the three separate *objects*. Our crossover matching algorithm maintains object identities throughout.

Figure 5 shows a case of severe fragmentation (which is not uncommon in challenging outdoor scenes where objects may have a wide range of sizes, and backgrounds are unconstrained). The bus is much larger than the other objects in this scene (car and pedestrian). Correspondingly, the gap between its fragments is comparable to the typical gap between distinct smaller objects. These fragments cannot be grouped using distance-based criteria alone. Our method correctly associates the fragments, while still tracking other objects separately.

Figure 6, from the PETS dataset, is even more challenging, and illustrates some limitations of our algorithm. *Objects* 464 and 465 are correctly detected, even though they enter as a *group*. An error occurs in the 4^{th} frame, where two persons (the referee and a red-shirted player) are labeled as *fragments* of *object* 352. This is because these 2 persons have similar motion for a few frames till one of them merges into another *group*. As a consequence, crossover matching for *group* 97 is also incorrect.

Figure 7 shows examples of how fragmentation might occur. The first 3 columns are examples in standard data sets from the CAVIAR project. In particular, the first two frames are sampled from a scene where the same person (#20) appears in a group and in fragments. The third column is from another scene where the person fragments because his shirt's color matches the background. The last 2 columns are from a street-side view. In one frame, we see a large gap between fragments of a person near the camera.

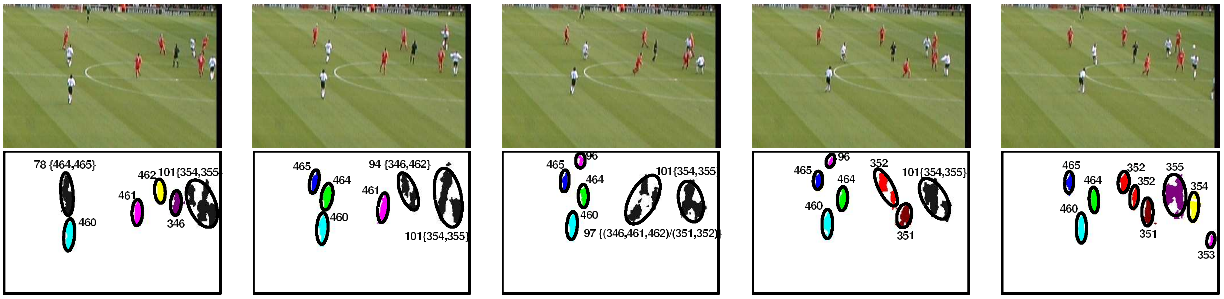


Figure 6. A more challenging data set (shown in same format as Fig. 4). See text for details.

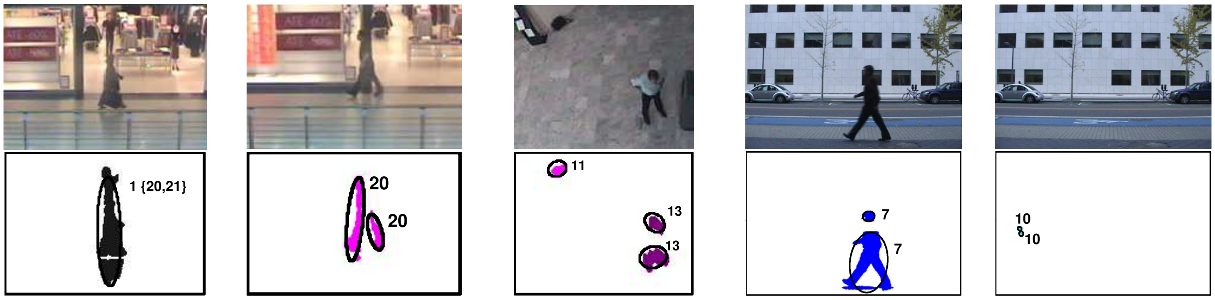


Figure 7. Some examples of fragmentation (shown in same format as Fig. 4). See text for details.

In the other frame, we see fragmentation of a different person due to partial occlusion behind a vehicle. Both of these are successfully labeled.

5. Conclusion

We have presented a framework for tracking a varying number of objects that might fragment or group, without employing class-specific models. The framework involves a generic object model that helps solve the fragment-object-group ambiguity. We maintain identity of objects across multiple interactions, which is essential for higher-level tasks. Our tracking algorithm currently works in batch, but can be made online, since, for each new frame, only some target-set tracks and inference graph vertices need to be updated. To do this, of course, one would store a sliding window of past frames.

Acknowledgements

The authors would like to thank Gerald Dalley, Kinh Tieu, Jason Williams, Chris Stauffer and Josh Migdal for thought-provoking discussions. Funding support was provided by DARPA.

References

[1] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.

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

[3] 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 *Proc. ACIVS*, 2003.

[4] G. Gennari and G. D. Hager. Probabilistic data association methods in visual tracking of groups. In *CVPR*, 2004.

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

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

[7] J. Marques, P. Jorge, A. Abrantes, and J. Lemos. Tracking groups of pedestrians in video sequences. In *IEEE Workshop on Multi-Object Tracking*, 2003.

[8] J. Migdal and W. E. L. Grimson. Background subtraction using markov thresholds. In *IEEE WMVC*, 2005.

[9] P. Nillius, J. Sullivan, and S. Carlsson. Multi-target tracking—linking identities using bayesian network inference. In *CVPR*, 2006.

[10] A. Pece. From cluster tracking to people counting. In *PETS*, 2002.

[11] A. Senior. Tracking people with probabilistic appearance models. In *PETS*, 2002.

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

[13] J. Sullivan and S. Carlsson. Tracking and labelling of interacting multiple targets. In *Proc. ECCV*, 2006.

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