A Context-Based Tracker Switching Framework*

Ambrish Tyagi James W. Davis
Dept. of Computer Science and Engineering
Ohio State University, Columbus, OH, USA
{tyagia,jwdavis}@cse.ohio-state.edu

Abstract

We present a robust framework for tracking people in crowded outdoor environments monitored by multiple cameras with a goal of real-time performance. Since no single algorithm is perfect for the task of object tracking in all cases, we instead take an alternate approach. Our algorithm dynamically switches between several available trackers on-the-fly by evaluating the current state/context of the scene. Autonomous agents that make the switching decisions are assigned to each object in the scene. Initialization of new agents and the handoff between various tracking algorithms are completely automated. The collaboration between different trackers is shown to improve performance compared to the individual methods in terms of both computation and reliability. The tracker switching framework is evaluated on a multi-camera dataset and both qualitative and quantitative results are presented.

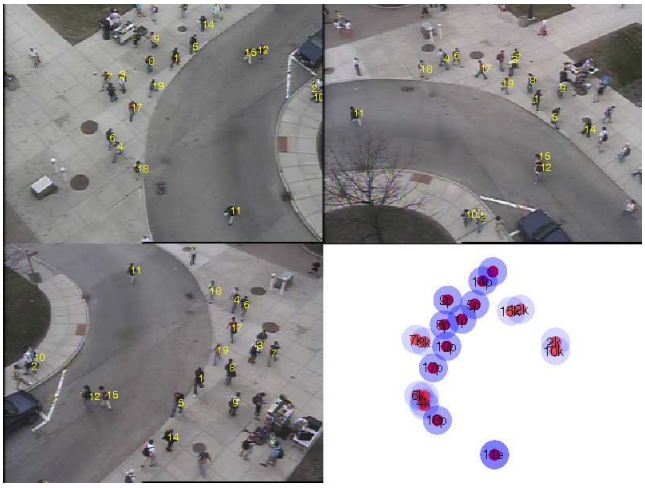


Figure 1. A busy outdoor scene monitored by three cameras. People seen in all camera views are tracked by agents that switch between available tracking methods based on the scene context. Numbers in yellow denote the unique agent IDs.

1. Introduction

Tracking multiple objects in a scene is a prerequisite for most higher-level dynamic computer vision tasks. A good tracking system should be able to run for long periods of time with the automatic capability to initialize new trackers, identify and replace failed trackers, and most importantly, operate efficiently.

Many algorithms exist for tracking objects in a scene, each possessing different performance characteristics. The usual approach in this situation is to manually select the algorithm which has the best average performance. However, this strategy has drawbacks since the selected algorithm may not be optimal as the scene evolves. The lack of existence of the perfect tracker motivates us to design a framework that switches between multiple component trackers to harnesses their robustness in particular contexts, while adhering to the goal of real-time tracking. The result is a net improvement in the fidelity of object tracking, even for complex, cluttered scenes with multiple people and occlusions.

Our approach is to employ multiple trackers that have complimentary modes of success (when in context) and failures (when out of context). Also, these tracking algorithms range from simple to complex in terms of computation. We do not always need to deploy the most expensive tracker in all cases. The simplest of these trackers is a Euclidean tracker that associates observations based only on the positional information. A predictive tracker additionally makes use of the object dynamics (*e.g.*, velocity) on top of the positional information. Finally a kernel-based tracker employs position, velocity, and appearance cues to track objects. The framework switches between these trackers based on the scene context.

In this paper we propose to switch between various tracking algorithms, as opposed to fusing the output of multiple algorithms. Firstly switching, *i.e.*, dynamically selecting

*This research was supported in part by the National Science Foundation under grant No. 0236653. Appears in IEEE Workshop on Motion and Video Computing (WMVC) 2008.

the appropriate algorithm, is a more computationally efficient approach of combining different tracking methods. Secondly, a fusion technique requires additional capabilities as part of the tracker to quantify/assess the tracker performance for (weighted) combination.

The proposed tracker switching framework is applicable to single view 2D tracking, but in this paper we will demonstrate its effectiveness for 3D tracking of *low resolution* people targets in a typical surveillance environment monitored by multiple synchronized cameras with overlapping views. In addition to the robustness afforded by multiple cameras, a calibrated system provides 3D trajectories of the people in the scene, which is desired in many applications (*e.g.*, 3D site monitoring).

The remainder of the paper is organized as follows. We review related work in Sect. 2. We present the main algorithm in Sect. 3 with details on features used for 3D tracking, the component trackers, and the tracker switching framework. An experimental evaluation of the framework is presented in Sect. 4 and we summarize and give concluding remarks in Sect. 5.

2. Related Work

Many 2D tracking algorithms have been proposed in the past (see survey in [14]). More recently there has been a surge of interest in 3D tracking algorithms [1, 4, 8, 13]. The idea of combining tracker outputs for robustness has been previously proposed (*e.g.*, [6, 10, 11]). In [6], the tracker outputs are fused in a probabilistic framework. An alternate fusion technique is proposed in [11] where the main tracking module consists of four parts including motion detection, region tracker, head detector and an Active Shape Tracker. All these modules are required to be executed as they assist each other to improve the tracker output. Similarly [10] combines the result of a region based correlation tracker with an adaptive contour tracker.

3. Algorithm

Our approach to object tracking is based on the idea of switching between various available tracking methods as needed. For a complex scene we can conceive tracking agents assigned to each object of interest that can switch between the various available tracking methods based on the current context of the scene. These agents evaluate the global scene context by making decisions locally (additionally affording a parallel implementation).

We select three component trackers to demonstrate the context-based switching framework. These tracking methods have complimentary success and failure modes. The simplest of the three uses only the positional information (of features) to track an object. This method is useful to track objects that exist in isolation. The second tracker adds another layer of complexity by additionally using the information about object dynamics. This tracker is applicable in situations with limited object interactions and it can also handle missing observations. Finally, an appearance tracker is introduced to disambiguate between targets involved in more complex interactions (*e.g.*, groups). Evidently, the computation requirements also increase as we move from a simpler to a more complex tracker.

Figure 2. Three camera views of a person (left) and the resulting point cloud using VH reconstruction (right).

The following description assumes that we have a metric calibration of the space observed by the cameras. The ground plane is assumed to be known and the 3D coordinate system is oriented such that the normal to the ground plane is aligned with the Z-axis. Notation-wise, lowercase letters like $\mathbf{x}$ represent points in 2D space and capital letters like $\mathbf{X}$ represent points in 3D space. Homogeneous coordinates in either space are represented with a tilde (*i.e.*, $\tilde{\mathbf{x}}$). Matrices $\mathbf{P}^i$ are used to project 3D points into the image plane (*i.e.*, $\tilde{\mathbf{x}}^i = \mathbf{P}^i\tilde{\mathbf{X}}$). Operator $||\bullet||$ denotes the L^2 norm.

3.1. Object Representation

Typical 2D point correspondence tracking algorithms extract foreground blobs and use the blob centroids as features to associate over time. Further analysis is done in order to deal with complex situations including blob merges, splits, and occlusions [7]. Similarly, for 3D tracking we extract 3D blobs and derive useful features to track. For a tracking application, the precise shape of an object is not important. Therefore instead of employing costly 3D matching/reconstruction methods to extract precise 3D shapes, we prefer to use a simple visual hull reconstruction method that gives us a coarse shape and location of objects present in the scene.

Visual hulls (VH) are a geometric entity that are often used in dealing with silhouette-based image understanding [5]. Here we will use VH to generate 3D point clouds of people to track. We begin by obtaining foreground silhouette images for each camera view using background subtraction [3]. Then, we voxelize the 3D space viewed by the multiple cameras. Let this voxelization be a 3D indexed array denoted by $\mathbf{H}$. The center of each voxel is projected into each foreground silhouette image. The number of cameras viewing a projected voxel center (indexed by u, v, w) is stored in $\mathbf{H}(u, v, w)$. Finally, we threshold $\mathbf{H}$ to obtain

the voxels that are viewed by all the cameras. The advantage of using VH is that the operation is constant time in the number of voxels. To speed up the point cloud generation, we do a two level coarse-to-fine VH reconstruction. First a coarse point cloud is generated using bigger voxels and then the result is refined by densely sampling the *neighborhood* of each 3D point obtained from the previous stage.

In Fig. 2 we show the result of VH reconstruction (right) obtained from the three views of a person (left). As seen in the figure, the shadow projected on the ground is also reconstructed. Given the 3D calibrated space and the knowledge of the ground plane, we can effectively deal with the shadow artifact by simply disregarding all points below a certain height.

The foreground reconstruction gives us the 3D point clouds of all the objects in the scene. In order to separate different objects we need to run a connected component algorithm to identify individual clusters. We create an undirected forest of graphs where nodes are the 2D projections of the point clouds on the ground plane (obtained by simply dropping the Z coordinate). Two nodes are connected by an edge if the Euclidean distance between them is less than d_g. Finally we use the depth first search algorithm to find the connected components in this forest. These connected components correspond to point cloud clusters of individual objects or groups of objects that are very close to each other. We store the clustering result and the corresponding cluster centers denoted by $\mathbf{C}_c^t$, where $c \in 1 \dots N_t$ and N_t is the number of clusters, at time t.

3.2. Component Trackers

We employ three trackers that range from simple to complex in terms of their tracking capabilities and computation.

3.2.1 Euclidean Tracker ($\mathcal{E}$-tracker)

The simplest of the three trackers is a Euclidean ($\mathcal{E}$) tracker that finds the association between the past location and new observation based only on the Euclidean distance. This tracker is efficient and it does not make assumptions about the object's dynamical model or appearance, although it assumes that an observation is always present. This is suited for situations where the objects being tracked are well separated and are not involved in any interactions with other objects or groups. Such simplicity obviously comes at a cost of fragility as the failure modes for this tracker arise due to object interactions, feature/observation noise, occlusions, merges and splits of VH results, *etc*.

The active context for the $\mathcal{E}$-tracker requires the object detections (VH results) to be "*reasonably*" far away from each other. Let X_i^t denote the 3D location of object i at time t. The new location for object i at time $t+1$ is given by, $\mathbf{X}_i^{t+1} = \mathbf{C}_j^{t+1}$, where $\mathbf{C}_j^{t+1}$ is the observation (cluster center) closest to the object's previous location. The observation $\mathbf{C}_j^{t+1}$ is marked as "accounted" for later analysis.

3.2.2 Predictive Tracker ($\mathcal{P}$-tracker)

The Predictive ($\mathcal{P}$) tracker layers another level of complexity to the previous tracker by employing the dynamics of the object in addition to its location. The additional information about the object dynamics can be used for target disambiguation in cases of "*medium level*" interaction between objects and predict object location in times of occlusions/missing observations. For our experiments we used a simple weighted averaged velocity model to predict target locations. More sophisticated predictive frameworks (*e.g.*, Kalman filter) can be used if desired. Therefore, our $\mathcal{P}$-tracker is an $\mathcal{E}$-tracker that additionally keeps a running weighted average of the object velocity, $\mathbf{V}_i^t$, for the last k frames,

$$\mathbf{V}_i^t = \frac{\sum_{p=t}^{t-k+2} p * (\mathbf{X}_i^p - \mathbf{X}_i^{p-1})}{\sum_{p=t}^{t-k+2} p} \tag{1}$$

The active context for $\mathcal{P}$-tracker includes scenarios where the tracking agent has other interacting objects in its proximity and the cases of missing observations. The new location of object i at time $t+1$ is given by,

$$\mathbf{X}_i^{t+1} = \begin{cases} \tilde{\mathbf{X}}_i^{t+1}, & \text{if missing observation} \\ \mathbf{C}_j^{t+1}, & \text{otherwise} \end{cases} \tag{2}$$

where $\tilde{\mathbf{X}}_i^{t+1} = \mathbf{X}_i^t + \mathbf{V}_i^t$ is the predicted object location, and $\mathbf{C}_j^{t+1}$ is nearest observation (if available) to $\tilde{\mathbf{X}}_i^{t+1}$ that can be assimilated (Sect. 3.3). The observation $\mathbf{C}_j^{t+1}$ is marked as "accounted" if it is used by the $\mathcal{P}$-tracker. Finally, the object velocity is updated at the end of the tracking iteration. In our framework, we only update the $x-$ and $y-$ component of velocity.

3.2.3 Appearance Tracker ($\mathcal{K}$-tracker)

Occasionally, objects being tracked come extremely close to each other due to interaction or high crowd density. In these situations simple position and motion cues are not sufficient to robustly track these objects and hence an additional layer of complexity (*e.g.*, appearance) is desired. We propose to use a Kernel ($\mathcal{K}$) based appearance tracker to track objects based on the color features in addition to position and velocity. An example of 2D kernel tracking is the popular mean-shift algorithm [2].

The basic 2D mean shift tracking algorithm can be extended to track a given object directly in the 3D space [13]. The new 3D location, $\mathbf{X}^{t+1}$, given the current position $\mathbf{X}^t$, can be estimated recursively from

$$\mathbf{X}^{t+1} = \frac{\sum_{\mathbf{Y} \in \mathcal{S}(\mathbf{X}^t)} \mathbf{Y} k'(\mathbf{X}^t - \mathbf{Y}) w(\mathbf{Y})}{\sum_{\mathbf{Y} \in \mathcal{S}(\mathbf{X}^t)} k'(\mathbf{X}^t - \mathbf{Y}) w(\mathbf{Y})}, \tag{3}$$

where $k(\cdot)$ is the profile of a smooth isotropic kernel, whose derivative is $k'(\cdot)$, and $w(\cdot)$ is a weighting function. The summation is performed in the 3D neighborhood of $\mathbf{X}^t$, *i.e.*, $\mathbf{Y} \in \mathcal{S}(\mathbf{X}^t)$. A proper sampling rate, s_{3d}, needs to be selected to discretize the 3D space. The choice of s_{3d} depends on how fine the features are spread in the 3D space.

Probability density functions $\hat{\mathbf{q}} = \{\hat{q}_u\}_{u=1,\ldots,m}$ and $\hat{\mathbf{p}}(\mathbf{X}) = \{\hat{p}_u(\mathbf{X})\}_{u=1,\ldots,m}$ of m-features are defined for the given *target model* and the *target candidate* at location $\mathbf{X}$, respectively. Target representations from 2D are modified to combine features from N sensors as

$$\hat{q}_u = C \sum_{i=1}^{N} \sum_{\mathbf{Y}^* \in \mathcal{S}(\mathbf{0})} R(\mathbf{P}^i \tilde{\mathbf{Y}}^*, u) k(\mathbf{Y}^*) , \quad (4)$$

$$\hat{p}_u(\mathbf{X}) = D \sum_{i=1}^{N} \sum_{\mathbf{Y} \in \mathcal{S}(\mathbf{X})} R(\mathbf{P}^i \tilde{\mathbf{Y}}, u) k(\mathbf{Y} - \mathbf{X}) , (5)$$

where C and D are constants chosen such that $\sum_{u=1}^{m} \hat{q}_u = 1$ and $\sum_{u=1}^{m} \hat{p}_u(\mathbf{X}) = 1$, respectively. For our experiments $m = 512$ as we use joint color histograms with 8 bins per channel. Furthermore, $R(\tilde{\mathbf{x}}, u) = \delta\left[b(\mathbf{x}) - u\right] \mathcal{V}(\mathbf{x})$, where the function $b : \mathcal{R}^2 \rightarrow \{1, \ldots, m\}$ associates the pixel at location $\mathbf{x}$ to its corresponding bin $b(\mathbf{x})$ in the quantized feature space, δ is the Kronecker delta function, and $\mathcal{V}(\mathbf{x})$ is a boolean function that evaluates to 1 if the point coordinates $\mathbf{x}$ are valid (*i.e.*, the point lies within the image view).

The weight function, $w(\mathbf{Y})$, is modified accordingly to accommodate contributions from all N camera views. A popular choice for the kernel function $k(\cdot)$ is the Epanechnikov kernel due to its simplicity and guarantee of convergence. The kernel bandwidth relates to the volume of the object being tracked. During initialization, a 3D volume defining the object of interest is selected. Since the real size of the object (such as a person) does not change over time, the volume containing it remains constant. This automatically solves the scale selection problem plaguing the standard 2D trackers since projection of this volume to individual views automatically selects the appropriate regions in the images. Also, the feature fusion is invariant to view changes, feature corruption, and occlusions, and the unified 3D tracker does not need to solve for object correspondence [13].

The $\mathcal{K}$-tracker like any other kernel tracker is good for short term target disambiguation, but, if used for long durations it is likely to drift off the object as the appearance models tend to change. Therefore, the active context in which this tracker should be used is when the object interacts closely with one or more objects, where the position and motion cues alone are not sufficient for tracking.

At each invocation of the $\mathcal{K}$-tracker, the local search is seeded from both the previous and the current predicted locations of the object. Of the two results, the one that is a better match in terms of the Bhattacharyya coefficient between the model and target location is selected. Finally, the obtained result is validated against the VH clusters by evaluating the volume overlap between the object kernel and the nearest cluster. If a sufficient overlap is found then the corresponding cluster observations is marked as "accounted".

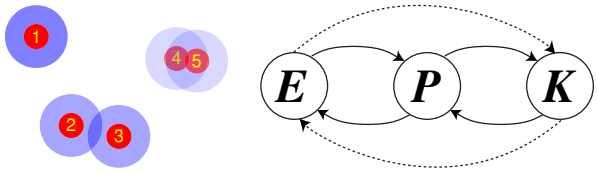
3.3. Tracker Switching Framework

We assign a tracking "agent" to each object in the scene. These agents are expected to operate under autonomous control, perceive their environment, and adapt to the changes by making rational decisions so as to achieve the best tracking outcome. In this section we describe the factors on which these agents base their actions about switching between available trackers. The tracking methods described in Sect. 3.2 have varying levels of complexity, therefore, the agents need to be able to make decisions on-the-fly about which of these methods are sufficient to track a particular object given the current scene configuration. When a simple tracker can suffice and produce robust tracking results then there is no need to invoke more complicated methods.

3.3.1 Local Context

We describe our tracker switching framework in context of the multi-camera outdoor surveillance domain. People and groups that traverse the scene constitute the objects of interest for tracking. An advantage of the calibrated 3D space is that we can get an estimate about the rough extent/size of the objects we are interested in tracking. With this knowledge we can define local "*spheres of influence*" centered on each person. Each person can be envisioned to have an imaginary *inner* and an *outer* sphere, or zone, of influence. The inner zone of influence is essentially the 3D extent of the person. The outer zone specifies the boundary outside which all non-interacting (thus unimportant) objects may exist. Let r and R denote the radius of inner and outer spheres, respectively. The outer zone of influence of an agent i is said to *interact* with another agent j if $||\mathbf{X}_i - \mathbf{X}_j|| \leq 2R$ and similarly the inner zone interacts if $||\mathbf{X}_i - \mathbf{X}_j|| \leq 2r$.

In Fig. 3(a) we show some possible configurations that may arise due to agent interaction. The 2D projection of the agent's zones of influence on the ground plane are shown. The $\mathcal{E}$-tracker is suitable to be used when the outer zone of influence of an agent does *not* interact with another agent (*e.g.*, object 1 in Fig. 3(a)). A $\mathcal{P}$-tracker is deemed more appropriate in case of a missing observation or if the outer zone (but *not* the inner) of an agent interacts with another agent (*e.g.*, objects 2 and 3 in Fig. 3(a)). Finally, if the agent gets too close to another agent, *i.e.*, their inner zone of influence overlap, then a $\mathcal{K}$-tracker should be used in order



(a) (b)

Figure 3. (a) Possible agent configurations. The spheres of influence are projected on the ground plane. (b) Finite state automaton representing the proposed tracker switching framework.

to avoid confusions by using the position, velocity, and appearance features jointly (*e.g.*, objects 4 and 5 in Fig. 3(a)).

Thus, an agent can switch between the three trackers depending on the level of interaction it is involved in. Our tracker switching framework can be viewed as a finite state automaton as shown in Fig. 3(b). We assume that an object can move a maximum distance of $2r$ between two consecutive frames. This is a reasonable assumption for various motions observed in the surveillance environment, including people riding bikes. Moreover, if this were not the case (*e.g.*, due to slower frame rate) then we can simply increase the extent of both inner and outer zones. Therefore, if we stipulate the condition, $R > 2r$, then an object outside the zones of influence of an agent cannot move too quickly as to penetrate the inner zone of this agent from one time instant to the next. Hence the transition from/to $\mathcal{E}$-tracker to/from $\mathcal{K}$-tracker is not expected to occur at a high video frame rate (hence shown as dashed arrows in Fig. 3(b)).

3.3.2 Switching Conditions

Subscribing to the aforementioned concepts, the conditions for switching from one tracker to another are as follows.

Switching from $\mathcal{E}$-tracker: For an agent i if the current method is $\mathcal{E}$-tracker and its outer zone interacts with another agent then a handover to $\mathcal{P}$-tracker is required. For the $\mathcal{E}$-tracker to successfully track an object we also need to ensure that a proper observation is always available since this method cannot deal with missing or ambiguous observations. We can imagine the observations in a new frame (at t+1) to have their own zones of influence. An observation whose inner zone interacts with the agent can be considered for assimilation (by that agent). Therefore, another precondition for $\mathcal{E}$-tracker is the presence of *exactly one* observation that can be assimilated by the agent, and it should also be the only observation whose outer zone interacts with the agent. Failure of this condition results in a switch to $\mathcal{P}$-tracker. The handoff process estimates the weighted average velocity from the history of previous tracked locations.

Switching from $\mathcal{P}$-tracker: If an agent currently employs $\mathcal{P}$-tracker then a potential handoff to either $\mathcal{E}$-tracker or $\mathcal{K}$-tracker is possible. The condition for switching back to the $\mathcal{E}$-tracker requires that no other agent's outer zone interacts with the given agent and also there is exactly one observation present in the agent's outer zone of influence that can be assimilated by the agent. In addition, the $\mathcal{P}$-tracker should have assimilated an observation in the previous frame, *i.e.*, it should not be doing predictive-only tracking (without assimilating any observations). On the other hand, if the given agent approaches near another agent such that their inner zones overlap then the agent should handover the tracking task to $\mathcal{K}$-tracker. Moreover, a handover to $\mathcal{K}$-tracker is also imperative if $\mathcal{P}$-tracker does predictive-only tracking for last T_P frames. During the handoff to the $\mathcal{K}$-tracker, the appearance model is learned from the last frame where an observation was successfully assimilated. Hence, we need to keep a short buffer of previous video frames.

Switching from $\mathcal{K}$-tracker: Finally if the current agent state is set to use the $\mathcal{K}$-tracker then it may switch back to $\mathcal{P}$-tracker. The handover back to $\mathcal{P}$-tracker is contingent on the condition that the agent's inner zone does *not* interact with any other agent. The handover to $\mathcal{P}$-tracker is not possible if the output of $\mathcal{K}$-tracker is not validated (*i.e.*, the output does not overlap with any VH cluster) and the previous transition from $\mathcal{P}$-tracker to $\mathcal{K}$-tracker was due to $\mathcal{P}$-tracker timeout (*i.e.*, predictive-only tracking for T_P frames).

3.3.3 Agent Initialization and Termination

A tracking agent is initialized for each cluster found in the VH results of the initial scene. For each subsequent time step we evaluate the local context of an agent and switch trackers, if required. The assigned tracker is used to detect the object's new location and the state of the corresponding agent is appropriately updated. Finally, the scene is re-analyzed, and new *temporary* agents are assigned for all "unaccounted" observations. The temporary agents that have survived for at least T_S frames are made permanent. New objects are only initialized when seen in all camera views.

During the tracking iteration if a temporary agent misses any observation, or attempts to assimilate an already accounted observation, then it is scheduled to terminate. Additionally, if a permanent agent seeks to assimilate an observation already claimed by a temporary agent, then the observation is assigned to the former and the temporary agent is terminated. Furthermore, we stop tracking an agent once it goes out of any camera view. Also, an agent doing predictive-only tracking using a $\mathcal{P}$-tracker (*i.e.*, case of missing observation) is terminated if they are closer to the edge of a camera view with their velocity vector pointing in the direction of the edge. Finally, for a permanent agent, if the output of $\mathcal{K}$-tracker is not validated for last T_K frames then the agent is scheduled to terminate.

4. Experiments

We recorded 30 different sequences from various locations of a busy University campus area monitored by three synchronized cameras (*e.g.*, Fig. 1). Each camera recorded 320×240 color images at 30Hz. The sequence lengths varied between 550–650 frames each. The auto-calibration approach of [9] was used to obtain a metric calibration space and derive the camera matrices $\mathbf{P}^i$. Ground truth was generated by manually annotating all trajectories of people in half of these sequences. People locations in all three views were marked at every 15 frames. The 2D image coordinates were triangulated to obtain the 3D coordinates and were interpolated to obtain the 3D ground truth object locations at each frame. We annotated $\sim$200 trajectories in this endeavor.

The goal of these experiments is to track the 3D location of all people in the scene, and to compare the proposed algorithm with individual tracking methods in terms of the tracking error w.r.t. the ground truth. The dataset contains several natural instances of crowding, occlusions, view changes, group interactions, *etc*. The typical person size in these images varied between 7×14 to 20×40 pixels.

For our experiments, the timeout durations were set to $T_P = T_K = T_S = 10$ frames. The radius of inner spheres, r, was set to 1.1 or 0.8 (for different locations), $R = 2.5r$, and $d_g = r/5$ (for clustering). For the $\mathcal{K}$-tracker, the bandwidth was automatically selected for each object using the point cloud clusters, and the corresponding sampling rate s_{3d} was selected to produce 20 samples in each dimension. Currently, we do not conduct detailed timing tests (algorithm prototypes implemented in Matlab). The intent of this study is to evaluate the tracking performance of our proposed approach and demonstrate the efficacy of switching.

4.1. Qualitative Results

Fig. 1 shows a snapshot of the tracking results of the automatic context-based switching framework on one of the sequences. Individual objects are tracked by agents that switch between the available component trackers. The schematic on the right-bottom shows the top-down view of the agents interacting with each other, overlaid by their unique ID and the current tracking method in use. The red and the blue disks are the projections of the agent's inner and outer zones of influence, respectively. The transparency of the disks correspond to the level of interaction (*e.g.*, agents extremely close to each other have high transparency).

In Fig. 4 we compare the output of our tracking system to the ground truth trajectories of the tracked objects. The projection of 3D trajectories on the ground plane are shown. These examples demonstrate the effectiveness of the tracking results generated by the framework.

Lastly, in Fig. 5 we show an example of interaction between two agents. Projection of the trajectories on the ground plane are shown. The colors of each segment represent the different tracking method being used. Agents 1 and 2 start moving in opposite directions (shown by red arrows). Initially, when they are isolated, they both employ the $\mathcal{E}$-tracker (briefly, agent 2 uses $\mathcal{P}$-tracker due to interaction with another agent not shown). As they move closer and their outer zones interact, both agents switch to the $\mathcal{P}$-tracker. Eventually the agents come in close proximity and their inner zones interact, thus requiring them to use the $\mathcal{K}$-tracker. Finally, they both move away from each other, switching back to $\mathcal{P}$-tracker and then the $\mathcal{E}$-tracker.

Figure 4. Output of the tracking framework (red) overlaid on top of the ground truth trajectories (thick, gray)

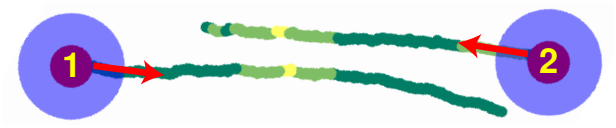


Figure 5. Switching between component trackers due to agent interaction. Segments shown in dark green, light green, and yellow denote the use of $\mathcal{E}$, $\mathcal{P}$, and $\mathcal{K}$-trackers, respectively.

4.2. Evaluation Metrics

To evaluate our multi-object tracker framework we use nine metrics proposed in [12] that capture the notion of *configuration* (*i.e.*, the number and location of objects and trackers in the scene) and *identification* (*i.e.*, the consistent labeling of objects over a long period of time). A good tracking system should track all objects by detecting their correct locations in each frame, and also should track individual objects consistently over long periods of time.

In this section, the 3D ground truth locations of the objects and the agent/tracker outputs are denoted by $\mathcal{G}$ and $\mathcal{T}$, respectively. A ground truth location $\mathcal{G}_i$ at time t is said

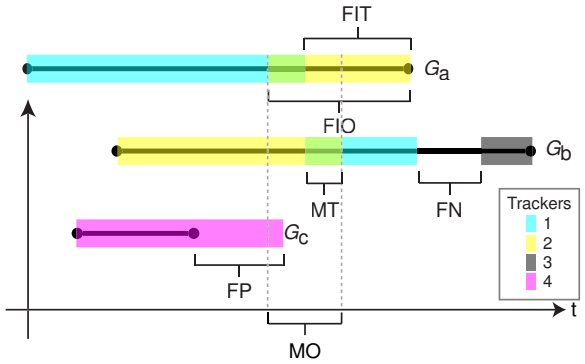


Figure 6. Representative errors in multi-object tracking. Ground truth objects $\mathcal{G}_{a,b,c}$ are tracked by trackers $\mathcal{T}_{1\ldots4}$. All error instances are not marked on the time-line. (Fig. looks good in color)

to be *associated* with the tracker output $\mathcal{T}_j$ at time t if the Euclidean distance $||\mathcal{G}_i - \mathcal{T}_j|| \leq r$.

To demonstrate the various kinds of tracker errors that can occur in multiple object tracking we present a simple schematic in Fig. 6 where the three objects $\mathcal{G}_{a,b,c}$ are tracked by four trackers $\mathcal{T}_{1\ldots4}$. A track $\mathcal{T}$ *identifies* the $\mathcal{G}$ it spends the most time tracking, and a $\mathcal{G}$ is *identified* by the $\mathcal{T}$ that tracks it for the longest duration. Hence, trackers $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$, and $\mathcal{T}_4$ identify ground truth objects $\mathcal{G}_a$, $\mathcal{G}_b$, $\mathcal{G}_b$, and $\mathcal{G}_c$, respectively. In the other direction, objects $\mathcal{G}_a$, $\mathcal{G}_b$, and $\mathcal{G}_c$ are identified by trackers $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_4$, respectively.

Trackers that do not associate with any ground truth objects result in false positive (FP) errors. Ground truth segments that are not tracked by any tracker contribute to false negative (FN) errors. Multiple tracker (MT) and multiple object (MO) errors occur when there are many-to-many associations between $\mathcal{G}$s and $\mathcal{T}$s. Notice how $\mathcal{T}_2$ is simultaneously associated with two objects $\mathcal{G}_a$ and $\mathcal{G}_b$ for some time duration, thus resulting in MO errors. Also, at one time both $\mathcal{T}_1$ and $\mathcal{T}_2$ are tracking $\mathcal{G}_b$, thus resulting in MT errors. A $\mathcal{G}$ tracked by some $\mathcal{T}$, which is not its identifying tracker results in a falsely identified tracker (FIT) error. Similarly, falsely identified object (FIO) error occurs when a $\mathcal{T}$ tracks some $\mathcal{G}$ that is not identified by this tracker. The FIT and FIO metrics capture the track fragmentation (multiple trackers for one object) and track merge (one tracker for multiple objects) errors, respectively.

Tracker purity (TP) and object purity (OP) measure the proportion of time for which a $\mathcal{T}$ tracks the identified $\mathcal{G}$, and the time for which a $\mathcal{G}$ is tracked by the identifying $\mathcal{T}$, respectively. Finally, configuration distance (CD) keeps a count of difference between the number of $\mathcal{T}$s and $\mathcal{G}$s at each frame.

Like any other error metric, it is desirable to have a low FP, FN, MT, MO, CD, FIT and FIO errors, whereas, the purity measures, TP and OP, should be high for a good tracking system. All errors are normalized (at each frame) by $\max\left(N_{\mathcal{G}}^t, 1\right)$, where $N_{\mathcal{G}}^t$ is the number of $\mathcal{G}$s in frame t. It should be noted that some of the errors metrics are correlated, *e.g.*, a tracker that is not associated with any ground truth object will contribute to FP errors and also reduce the average TP.

4.3. Quantitative Results

To evaluate the performance of the proposed tracker switching framework we compared it with other possible tracker combinations that can be conceived from the given component trackers. Selecting two out of three trackers resulted in methods $\mathcal{E}$+$\mathcal{P}$ (*i.e.*, an agent can switch between an $\mathcal{E}$-tracker and a $\mathcal{P}$-tracker) and $\mathcal{P}$+$\mathcal{K}$. By construction since the switching between an $\mathcal{E}$-tracker and a $\mathcal{K}$-tracker is unexpected, we do not consider their combination in this comparison. Finally the three trackers can be used independently thus resulting in three more methods, namely $\mathcal{E}$, $\mathcal{P}$, and $\mathcal{K}$. Our proposed tracker switching method is denoted by $\mathcal{E}$+$\mathcal{P}$+$\mathcal{K}$.

We compared the aforementioned methods against each other based on the metrics (defined earlier) computed for all sequences for which ground truth is available. The results are shown in graphs of Fig. 7. Each bar shows the metric value averaged over all the frames of sequences used. The results for method $\mathcal{E}$ are omitted from the graphs since this simple tracker produced very high errors and low purity values, as expected. As seen from the plots, methods $\mathcal{E}$+$\mathcal{P}$+$\mathcal{K}$ and $\mathcal{P}$+$\mathcal{K}$ demonstrate better overall tracking performance compared to other methods.

Method $\mathcal{K}$ performs poorly on the FP, FN, FIT, TP, and OP metrics. The reason for its poor performance can be attributed to the susceptibility of the kernel tracker to drift over time. $\mathcal{K}$-tracker is robust for short term track disambiguation but, if used alone, it will drift off the object eventually, resulting in high FP/FN errors. This also leads to track fragmentation since multiple trackers are required to track a single ground truth object, and hence a higher FIT error. $\mathcal{K}$-trackers are mostly active for short durations and hence there is a lesser chance of track merges, thus a low FIO error is obtained.

Methods $\mathcal{P}$ and $\mathcal{E}$+$\mathcal{P}$ lack in performance primarily for the MT, MO, CD, FIT, and FIO metrics. Due to the missing $\mathcal{K}$-tracker in these methods, they fail to disambiguate objects as they become part of group interactions (3D blobs merges). Thus multiple trackers latch onto single clusters (group) and new trackers need to be started as the objects get separated eventually. The multiple trackers from before still incorrectly track single objects after blob splits. This also results in a higher number of agents in the scene. The tracker/object mismatch in crowded scenes therefore results in high value of these error metrics.

Finally, both the $\mathcal{P}$+$\mathcal{K}$ and $\mathcal{E}$+$\mathcal{P}$+$\mathcal{K}$ methods perform

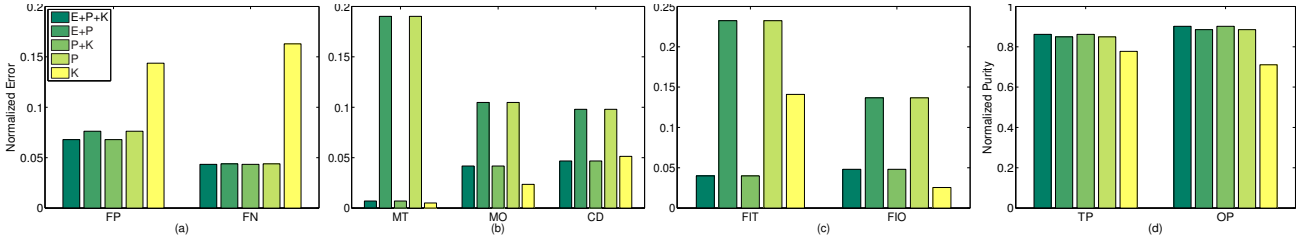


Figure 7. Comparison of tracking methods evaluated on metrics: (a) FP/FN (b) MT/MO/CD (c) FIT/FIO (d) TP/OP. A good tracker should have low errors for (a), (b), and (c), and high purity for (d). The legend shown in (a) holds for all the graphs.

	$\mathcal{E}$+$\mathcal{P}$+$\mathcal{K}$	$\mathcal{E}$+$\mathcal{P}$	$\mathcal{P}$+$\mathcal{K}$	$\mathcal{P}$	$\mathcal{K}$
$\mathcal{E}$	21348	23805	-	-	-
$\mathcal{P}$	6714	10923	28068	34728	-
$\mathcal{K}$	3772	-	3766	-	30914

Table 1. The number of times each individual tracker (rows) is invoked by various methods (columns).

similarly on all metrics. This begs the question: What benefit, if any, is achieved by using the $\mathcal{E}$-tracker? The primary advantage is the saving in computation time obtained by using this simple tracker in the situations when it is applicable. To quantify the computational savings we estimated the ratio of average times spent in one iteration of each of the $\mathcal{E}$,$\mathcal{P}$,$\mathcal{K}$-trackers, which turned out to be $1:3:60$. The number of times each of the three trackers were invoked by each method is shown in Table 1. From this information we obtain a relative improvement of 13.66% in terms of computation time (*e.g.*, it translates to a running time of 13 sec for $\mathcal{E}$+$\mathcal{P}$+$\mathcal{K}$ *vs*. 15 sec for $\mathcal{P}$+$\mathcal{K}$). Furthermore, the $\mathcal{E}$-tracker is additionally robust in situations when a fast moving isolated object suddenly changes its motion direction, and the $\mathcal{P}$-tracker fails due to incorrect velocity prediction.

5. Summary and Conclusion

We presented a context-based tracker switching framework that robustly tracks objects in a busy scene. A switching framework provides a practical way to combine multiple tracking algorithms. The framework draws upon the robust and salient properties of each component tracker by dynamically selecting the appropriate method based on the current scene context. We demonstrated our approach for tracking people in 3D using multiple cameras. The proposed algorithm was compared to individual component trackers and found to be more reliable and computationally efficient. In the future work, we plan to formally compare our switching framework to alternate techniques for tracker combination, such as fusion (both decision and feature fusion).

References

[1] J. Black and T. J. Ellis. Multi camera image tracking. *Image and Vision Comp.*, 24(11):1256–1267, 2006.

[2] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Trans. Patt. Analy. and Mach. Intell.*, 25(5):564–577, May 2003.

[3] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis. Real-time foreground-background segmentation using codebook model. *Elsevier Real-Time Imaging*, 11(3):172–185, June 2005.

[4] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera multi-person tracking for EasyLiving. In *Proc. Work. Vis. Surveillance*, pages 3–10, 2000.

[5] A. Laurentini. How far 3D shapes can be understood from 2D silhouettes. *IEEE Trans. Patt. Analy. and Mach. Intell.*, 17(2):188–195, 1995.

[6] I. Leichter, M. Lindenbaum, and E. Rivlin. A general framework for combining visual trackers – the "Black Boxes" approach. *Int. J. of Comp. Vis.*, 67(3):343–363, 2006.

[7] O. Masoud and N. Papanikolopoulos. A novel method for tracking and counting pedestrians in real-time using a single camera. *IEEE Trans. on Vehicular Tech.*, 50(5):1267–1278, 2001.

[8] A. Mittal and L. Davis. M2Tracker: A multi-view approach to segmenting and tracking people in a cluttered scene using region-based stereo. In *Proc. European Conf. Comp. Vis.*, pages 18–36, 2002.

[9] M. Pollefeys, L. V. Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *Int. J. of Comp. Vis.*, 59(3):207–232, 2004.

[10] K. Shearer, S. Venkatesh, and K. Wong. Combining multiple tracking algorithms for improved general performance. *Pattern Recognition*, 34(6):1257–1269, 2001.

[11] N. Siebel and S.Maybank. Fusion of multiple tracking algorithms for robust people tracking. In *Proc. European Conf. Comp. Vis.*, pages 373 – 387, 2002.

[12] K. Smith, D. Gatica-Perez, J. Odobez, and S. Ba. Evaluating multi-object tracking. In *Wkshp. on Empirical Evaluation Methods in Comp. Vis.*, volume 3, pages 36–36, 2005.

[13] A. Tyagi, G. Potamianos, J. Davis, and S. Chu. Fusion of multiple camera views for kernel-based 3D tracking. In *Proc. Wkshp. Motion and Video Computing*, pages 1–8, 2007.

[14] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4), 2006.