Kernel-Based 3D Tracking

Ambrish Tyagi[1], Mark Keck[1], James W. Davis[1], Gerasimos Potamianos[2]
[1] Dept. of Computer Science and Engineering, Ohio State University, Colombus, OH–43210
[2] IBM T.J. Watson Research Center, Yorktown Heights, NY–10598
Emails: {tyagia,keck,jwdavis}@cse.ohio-state.edu, gpotam@us.ibm.com

Abstract

We present a computer vision system for robust object tracking in 3D by combining evidence from multiple calibrated cameras. This kernel-based 3D tracker is automatically bootstrapped by constructing 3D point clouds. These points clouds are then clustered and used to initialize the trackers and validate their performance. The framework describes a complete tracking system that fuses appearance features from all available camera sensors and is capable of automatic initialization and drift detection. Its elegance resides in its inherent ability to handle problems encountered by various 2D trackers, including scale selection, occlusion, view-dependence, and correspondence across views. Tracking results for an indoor smart room and a multi-camera outdoor surveillance scenario are presented. We demonstrate the effectiveness of this unified approach by comparing its performance to a baseline 3D tracker that fuses results of independent 2D trackers, as well as comparing the re-initialization results to known ground truth.

1. Introduction

A *complex scene* by definition is one in which several moving objects (people, animals, vehicles) are present. In the surveillance domain, the ultimate goal is to monitor such a scene for an arbitrarily long period of time, detecting objects, recognizing the actions of these detected objects, and exposing anomalous behavior.

Embedded in the link between the first two processes in this surveillance pipeline is tracking. The problem of tracking can be loosely defined as the temporal association of objects. Many attempts to solve this problem have been proposed. Traditionally filtering techniques like Kalman filters and particle filters (as well as their variants) have been employed to address the problem, and more recently kernel-based techniques for tracking objects [5] gained approval due to their succinct description and computational efficiency.

These algorithms, although well tested in the image plane (2D), have rarely been extended to work in 3D. As multiple camera systems and data have become more affordable and available, the desire to track an object in 3D has surfaced. Some 3D tracking approaches do exist [2, 16, 17], and these focus on fusing results from independent 2D trackers to obtain 3D trajectories. This decision level fusion therefore suffers from the limitations present in 2D tracking.

In this paper we detail a fully automatic approach to kernel-based 3D tracking. We propose a kernel-based 3D tracker that inherently handles the limitations associated with 2D kernel-based trackers, namely scale-selection, occlusion, view-dependence, and correspondence across views. The 3D tracker handles these problems elegantly by fusing information at the *feature* level as opposed to the *decision* level. We strengthen this approach by introducing a method for automatic re-initialization and drift detection for the tracker. The method produces a set of 3D point clouds and clusters them to facilitate re-initialization. When drift in the model is detected, the tracker is automatically re-initialized using the centroids of the clusters. Results are shown on multiple datasets (both indoors and outdoors) and are compared with ground truth.

The remainder of the paper is organized as follows. We review related work in Sect. 2. We then describe our kernel-based 3D tracker in Sect. 3 and automatic re-initialization method in Sect. 4. An extensive experimental evaluation of the framework is presented in Sect. 5 and we summarize and give concluding remarks in Sect. 6.

2. Related Work

In the past, filtering and data association techniques have shown some success in the tracking domain. The simplest of these models, Kalman filters, assume that the underlying process driving an object is linear and Gaussian distributed. Extended Kalman filters (EKFs) have also been used, which allow the underlying process to be nonlinear. An even more general filtering method, particle filters, allow objects driven by a nonlinear process with non-Gaussian distribution to be tracked robustly. Beyond filter-

1-4244-1180-7/07/$25.00 ©2007 IEEE

ing methods, kernel-based methods have gained favor recently. Mean shift tracking [5] has seen wide use since its inception. A detailed survey on object tracking can be found in [15].

As stated in Sect. 1, these algorithms have mostly focused on 2D tracking (in the image plane). However, some recent systems have attempted to fuse information from different sensors to obtain trajectories in 3D. In [17] shape features are extracted from blobs and tracked in different views independently, and finally fused into 3D trajectories using an EKF. Similarly, [2] uses epipolar geometry to match centroids of foreground objects in multiple views. The points are triangulated, and the 3D points are then smoothed with a Kalman filter. In [16] a 2D mean shift tracking system was presented that was bootstrapped using motion cues and face detection. The 2D results were triangulated to obtain a 3D result.

Few other previous works attempt to address the problem of 3D tracking in special cases (mostly indoors). Both [6, 10] use stereo rigs to segment people moving in a closed room. The stereo setups, in addition to being expensive, are not suitable to track objects (*e.g.* people) at low resolution in say an outdoor surveillance scenario. Techniques like [8, 12] are also tailored to track high resolution targets and additionally rely on 3D body models.

In all of these cases the 3D trackers are constrained by either the limitations of the 2D trackers since the fusion of sensors is done at the decision level or they are tuned to work in limited conditions. Our proposed approach to tracking addresses these issues by tracking the desired object directly in the 3D space and is general enough to be applicable in various domains. The approach is a 3D generalization of 2D mean shift tracking, and will be outlined in Sect. 3. Furthermore in Sect. 4 we provide a mechanism by which the 3D mean shift algorithm will be able to automatically re-initialize itself. We do this by generating 3D point clouds in the scene and bootstrapping the tracker with these clouds when necessary, thus providing a fully automatic framework for 3D tracking.

3. Kernel-Based Tracking in 3D

In this section, we describe our approach for kernel-based 3D object tracking. The algorithm uses a feature level fusion framework to track the object directly in the 3D space. We explain the differences between the original 2D mean shift tracking approach and the new method, and highlight the advantages of this framework versus a decision level fusion approach.

In the remainder of the paper, we will use the following notation. Lowercase letters like $\mathbf{x}$ represent points in 2D space, whereas 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 $\mathbf{P}^i$ represent camera projection matrices for each view that project 3D points into the image plane (*i.e.*, $\tilde{\mathbf{x}}^i = \mathbf{P}^i\tilde{\mathbf{X}}$). Images will be denoted with a calligraphic font like $\mathcal{I}$.

3.1. Baseline 3D Tracking Using Decision Fusion

The simplest approach to 3D kernel-based tracking that comes to mind is to first track independently in each view, and then at each frame solve the correspondence problem and associate objects. The final 3D trajectories are obtained by triangulating these correspondences at each frame. Decision level approaches like this one have been adopted in [16, 17] and we will compare our proposed tracking algorithm to a similar *baseline* approach in which the 2D mean shift tracking results are combined to generate the 3D output. However, the shortcomings of this type of approach are apparent. Imagine for instance that a person's head is being tracked. Suppose that at some point in the sequence the person's head rotates. This type of feature corruption, as well as occlusion and other changes in view will result in failure of view-based 2D trackers and hence the 3D tracker.

3.2. Proposed Feature Level Fusion for 3D Tracking

A more principled approach than the one described above would be to track the object directly in the 3D space by obtaining features jointly from the contributing views. This feature fusion is invariant to view changes, feature corruption, and occlusions. Also, a unified 3D tracker eliminates the inconsistency in the 2D image locations as they are simply the projection of the 3D object location.

Here we will extend the basic 2D mean shift tracking algorithm to track a given object directly in the 3D space. The new 3D location, $\mathbf{X}_{t+1}$, given the current position $\mathbf{X}_t$, can be estimated recursively from

$$\mathbf{X}_{t+1} = \frac{\sum_{\mathbf{Y}\in\mathcal{S}(\mathbf{X}_t)} \mathbf{Y}k'(\mathbf{X}_t - \mathbf{Y})w(\mathbf{Y})}{\sum_{\mathbf{Y}\in\mathcal{S}(\mathbf{X}_t)} k'(\mathbf{X}_t - \mathbf{Y})w(\mathbf{Y})} \ , \qquad (1)$$

where $k(\cdot)$ is the profile of a smooth isotropic kernel, whose derivative is $k'(\cdot)$, and $w(\cdot)$ is a weighting function. Originally, in the 2D case, the summation was performed in the 2D pixel neighborhood. The summation is now performed in the 3D neighborhood of $\mathbf{X}_t$, *i.e.* $\mathbf{Y} \in \mathcal{S}(\mathbf{X}_t)$. Unlike in 2D, where the space is already discretized into pixels, one needs to select a proper sampling rate, s_{3d}, of the 3D region for the summation to be defined in the neighborhood $\mathcal{S}$. This will depend on how fine the features are spread in the 3D space.

We define the 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 for the given *target model* and the *target candidate* at location $\mathbf{X}$, respectively. We modify the target representations from 2D to combine features from N sensors as

$$\hat{q}_u = C \sum_{i=1}^{N} \sum_{\mathbf{Y}^* \in \mathcal{S}(\mathbf{0})} R(\mathbf{P}^i \tilde{\mathbf{Y}}^*, u) k(\mathbf{Y}^*) \,, \quad (2)$$

$$\hat{p}_u(\mathbf{X}) = D \sum_{i=1}^{N} \sum_{\mathbf{Y} \in \mathcal{S}(\mathbf{X})} R(\mathbf{P}^i \tilde{\mathbf{Y}}, u) k(\mathbf{Y} - \mathbf{X}) \,, (3)$$

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,

$$R(\tilde{\mathbf{x}}, u) = \delta\left[b(\mathbf{x}) - u\right] \mathcal{V}(\mathbf{x}) \,, \quad (4)$$

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 is modified to accommodate contributions from all N camera views such that

$$w(\mathbf{X}) = \sum_{i=1}^{N} \sum_{u=1}^{m} R(\mathbf{P}^i \tilde{\mathbf{X}}, u) \sqrt{\frac{\hat{q}_u}{\hat{p}_u(\mathbf{X})}} \,. \quad (5)$$

In contrast to the 2D case, where the weighted average of the pixel locations guide the search for the new location, the proposed algorithm instead operates in 3D, where each (discrete) point in space has a weight associated with it. The spatial 3D patches that are similar to the target model in the chosen feature space and are closer to the kernel center will have a higher weight. Note that Eqn. 1 now represents a weighted average of 3D locations.

A popular choice for the kernel function $k(\cdot)$ is the Epanechnikov kernel due to its simplicity and guarantee of convergence. It has the following profile

$$k(\mathbf{x}) = \begin{cases} \frac{d+2}{2h^2 c_d}(h^2 - \mathbf{x}^T\mathbf{x}) \,, & \text{if } \mathbf{x}^T\mathbf{x} \leq h^2 \\ 0 \,, & \text{otherwise} \end{cases} \quad (6)$$

where d is the number of dimensions, and c_d is a constant equal to the volume of a d-dimensional unit sphere. The bandwidth $h = h_{3d}$ relates to the volume of the object being tracked. Unlike the 2D bandwidth h_{2d} that changes with time (the scale selection problem), making adaptation in 2D a nontrivial process, the 3D bandwidth, once defined, remains constant for objects of fixed size. This is advantageous because after selecting the initial value for h_{3d} and s_{3d}, one does not need to worry about updating them during tracking.

3.3. Benefits of Feature Level Fusion

The salient features of the unified 3D tracking approach (Sect. 3.2) justifies its preference over the baseline decision level fusion algorithm. Firstly the proposed 3D algorithm intrinsically solves the problem of correspondence across views since we obtain the object location in each view by projecting the 3D tracker location into each image via $\{\mathbf{P}^i\}$ (unlike [2, 16]). A novel contribution of our proposed framework is that it also inherently handles the problem of target scale selection associated with 2D trackers. In past there have been attempts [3, 5] to adapt the 2D bandwidth over time, but there exists no optimal solution for the scale selection. During initialization, a 3D volume defining the object of interest is selected. Since the real size of the object (such as a human head) does not change over time, the volume containing it remains constant. The projection of this volume to individual views automatically selects the appropriate regions in the images. Hence, the problem of selecting the scale in each view, and at each iteration, is eliminated altogether. Moreover, as opposed to one mean shift iteration *per view* for the 2D algorithm, the 3D method requires only one unified mean shift iteration per time step. Also, extension to multi-object tracker is straightforward since we can start independent trackers for each individual object. Finally, any enhancements that can be applied to the 2D algorithm such as background weighted histograms [5], Bayesian filters, etc., can be also applied to the 3D case.

4. Automatic Re-initialization

Although the 3D tracking approach outlined in Sect. 3.2 is able to elegantly address the issues of scale selection, occlusion, view-dependence, and correspondence across views, it does not address the issues of re-initialization and drift detection. These issues are important for any practical system, as the automatic tracker should run for an arbitrarily long period of time with minimal human intervention. Therefore, we wish to strengthen our approach by incorporating a fully automatic re-initialization scheme based on clustering 3D point clouds.

Automatic initialization in 3D tracking has been attempted before in [16] where a complex face detection algorithm detected tracker drift, and then bootstrapped the tracker if required. This face detection algorithm, however, requires training, and therefore is dependent on the domain (dataset). To circumvent this problem, we propose a domain independent framework to bootstrap the tracker.

We start our algorithm by developing a background model for each view independently. We used a simple median background model for the results shown, but an adaptive model such as [9] could be employed. Next we perform background subtraction on each incoming frame $\mathcal{I}_t^k$ from camera k at time t and threshold this result to obtain the foreground images $\mathcal{F}_t^k$,

$$\mathcal{F}_t^k = \left|\mathcal{I}_t^k - \mathcal{B}^k\right| > T^k \quad (7)$$

where $\mathcal{B}^k$ and T^k are the background model and threshold value for the k^{th} camera, respectively.

Once foreground images are obtained, 3D point clouds are generated from them. We cover two methods to generate the 3D point clouds in Sections 4.1 and 4.2. We cluster these point clouds using mean shift clustering [4] and extract the centroids (denoted as set $\{\mathbf{Z}^j\}$) of the clusters to use as input to our automatic re-initialization framework.

We incorporate the centroids $\{\mathbf{Z}^j\}$ into our 3D tracking framework in the following manner. At every f_d-th frame we re-evaluate tracker performance by calculating the distance between each centroid $\mathbf{Z}^j$ and the current tracker output using a suitable distance metric. As we use a slightly different metric for indoor and outdoor experiments, we will describe them later in Sect. 5.1 and Sect. 5.2. Thus, for each centroid $\mathbf{Z}^j$ we compute a distance d_j. We then find the centroid $\mathbf{Z}^{min}$ that has the smallest distance value, d_{min}. If d_{min} is below our drift threshold, we assume the tracker is performing correctly and take no action. If d_{min} exceeds our drift threshold, we re-initialize the model using centroid $\mathbf{Z}^{min}$ as its new position.

In the following sections we present two methods for point cloud generation. The first method based on the visual hull algorithm has the advantage of being computationally more efficient as it has a constant overhead with respect to the 3D voxelization of the space. Additionally, we developed another method employing a robust matching algorithm based on epipolar geometry. This method can generate more accurate and detailed point clouds than a visual hull at a slightly higher computational cost.

4.1. Visual Hulls

Visual hulls are a geometric entity that are often used in dealing with silhouette-based image understanding [11]. Here we will use visual hulls to generate 3D point clouds for clustering. The algorithm is straightforward. First, we voxelize the 3D space being viewed by the multiple cameras. Let this voxelization be a 3D indexed array denoted by $\mathbf{C}$. Now, using the camera matrices, the center of each voxel is projected into each foreground image. The number of cameras viewing this projected point (indexed by u, v, w) is stored in $\mathbf{C}(u, v, w)$. Finally, we threshold $\mathbf{C}$ with the minimum number of cameras the system requires an object to be seen in (*e.g.*, we use a threshold of 2 to deal with noise in background subtraction).

The advantage of using visual hulls is that the operation is constant time in the number of voxels, which is defined by the sampling parameter s_{vh}. This parameter represents a tradeoff between speed of the algorithm and the granularity of the resulting hull/point cloud.

4.2. Epipolar Matching

If we desire to generate point clouds that capture the object shape more accurately, and the computational time

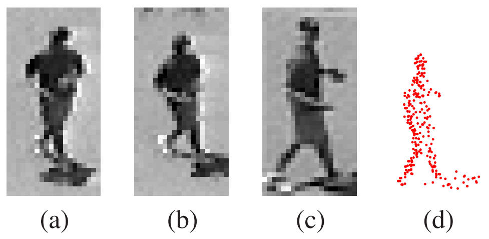

Figure 1. Three camera epipolar search. (a)-(c) Camera views of a person walking (20 × 40 pixels). (d) The resulting 3D point cloud reconstructed from matches found using the three camera epipolar search (Sect. 4.2).

3D Tracker	Min OE	Max OE	Avg. # of Re-init.
Baseline	93	157	14.54
Proposed	60	74	4.46

Table 1. Results showing superior performance of our proposed 3D tracker versus a tracker based on decision level fusion of independent 2D trackers. Minimum and maximum overall error (OE) in millimeter units.

is not a constraint (*e.g.* offline processing), then a method based on epipolar matching can be used. Here we present our version of *robust* epipolar matching designed specifically for three cameras. For the following description, let $\mathbf{F}_{ij}$ be the fundamental matrix between two cameras i and j such that $\tilde{\mathbf{x}}_i \mathbf{F}_{ij} \tilde{\mathbf{x}}_j = 0$, where $(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ are corresponding homogeneous points in the two camera views. We derive these fundamental matrices from the trifocal tensor [7].

Normal epipolar search [7] entails checking each pixel along epipolar line $\mathbf{l}_i = \mathbf{F}_{ij}\tilde{\mathbf{x}}_j$ and finding the best match for $\tilde{\mathbf{x}}_j$ based on some feature match, often color. This technique restricts search to only one dimension instead of two, but still leaves some ambiguity in the correspondence, and can result in incorrect matching.

To strengthen normal epipolar search, we use a third camera in the following way. We start in image $\mathcal{I}^i$ and choose a point of interest, $\tilde{\mathbf{x}}_i$. We find its corresponding epipolar line $\mathbf{l}_j = \mathbf{F}_{ji}\tilde{\mathbf{x}}_i$ in image $\mathcal{I}^j$ and search along this line for all possible matches. For each candidate match $\tilde{\mathbf{x}}_j$ along $\mathbf{l}_j$, we compute its corresponding line $\mathbf{l}_k = \mathbf{F}_{kj}\tilde{\mathbf{x}}_j$ in the third view $\mathcal{I}^k$ and find all candidate matches $\tilde{\mathbf{x}}_k$ along this line. For all candidate match pairs $(\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_k)$ the two lines $\mathbf{l}'_j = \mathbf{F}_{ij}\tilde{\mathbf{x}}_j$ and $\mathbf{l}'_k = \mathbf{F}_{ik}\tilde{\mathbf{x}}_k$ are computed in $\mathcal{I}^i$. The cross product of these two lines is their point of intersection, $\tilde{\mathbf{x}}'_i = \mathbf{l}'_j \times \mathbf{l}'_k$ in $\mathcal{I}^i$. If this point $\tilde{\mathbf{x}}'_i = \tilde{\mathbf{x}}_i$, then the three points are in correspondence.

In our particular case, we take as input three foreground silhouette images, and in the first image take only the contour pixels of the silhouette(s). We then use those as our set $\{\tilde{\mathbf{x}}_i\}$ and search through the other two images, finding the two candidate matches that project closest to $\tilde{\mathbf{x}}_i$. The

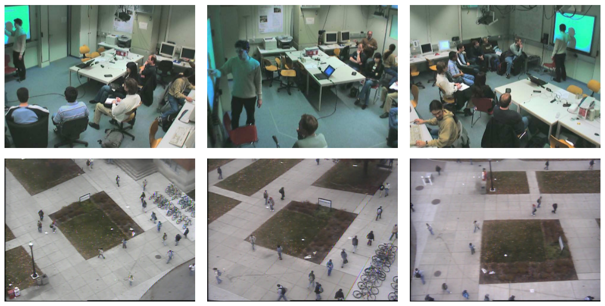
Figure 2. Top row: Sample views from three cameras of the CHIL (indoor) dataset. Bottom row: Three views from the wide-baseline outdoor surveillance scenario.

process is then repeated using the second image as the start of the search, and finally the third as well. The final set of corresponding triples is obtained by union of the three search results. Each triple is then triangulated to obtain a 3D point. When an object cannot be seen in all three views (*i.e.*, no point match pair projects near $\tilde{\mathbf{x}}_i$), we revert to normal epipolar search by simply matching color histograms to find the best possible match from the two views.

This three camera method of epipolar search has the advantage of delivering very reliable matches by eliminating the ambiguity present in simple epipolar search. This allows for the shape of a foreground object to be very detailed, given the resolution of the object in the image. An example of three images of a foreground object is shown in Fig. 1(a)-(c) and the resultant point cloud using our method is shown in Fig. 1(d). Unlike visual hulls, we are able to capture the concavities of the object and also overcome the problems encountered due to voxelization. However, as mentioned earlier, in cases where limited computational resources are available, a coarser (hierarchical) visual hull point cloud generation method may be preferred.

5. Experiments

In the following sections we report results on two different multi-camera scenarios. First in Sect. 5.1 we demonstrate our algorithm on a large dataset of annotated video. This set of experiments exhibits the effectiveness of the feature level fusion algorithm as opposed to the decision level fusion algorithm, and lays out how the automatic re-initialization framework can perform comparably with ground truth re-initialization.

However, due to the limitations of this dataset which are discussed in Sect. 5.1 and to demonstrate the effectiveness of our algorithm on more diverse data we collected a new dataset on which we report results in Sect. 5.2. We show again that feature level fusion outperforms decision level fusion, and again assert that the automatic re-initialization performs similarly to ground truth re-initialization.

In the remaining part of this paper, we will refer to the 3D tracking algorithm described in Sect. 3.1 as the *baseline* algorithm and compare its performance to the *proposed* algorithm of Sect. 3.2. The input to both these algorithms are the different (calibrated) camera views and the desired output is the 3D spatial location of the objects being tracked.

5.1. Seminar Room Database

We first evaluate the tracking algorithms on a number of "interactive seminar" video sequences recorded and *manually* annotated as part of the CHIL ("Computers in Human Interaction Loop") project [1]. The dataset consists of 26 video sequences – 2 recorded at the smart room in the Istituto Trentino di Cultura (ITC), Italy, and the remaining 24 inside the smart room of the University of Karlsruhe (UKA), Germany. Each sequence depicts a speaker giving a lecture to a small audience in the smart room. Each video segment contains approximately 4500 frames recorded at 15 Hz from 4 synchronized and calibrated cameras ($\sim$ 468k frames), although for the experiments with automatic re-initialization we only use the first three cameras. Images are captured at a 640×480 and 800×600 pixel resolution for the UKA and ITC sequences, respectively. The camera calibration information (both intrinsic and extrinsic) is made available with the dataset, from which we derive the camera matrices $\mathbf{P}^i$. The calibrated space corresponds to the actual distances (in mm) in the real world. The top row of Fig. 2 shows representative camera views from the dataset.

The goal of this experiment is to track the speaker's head in 3D through each sequence in an effort to compare the baseline and the proposed trackers, and to compare ground truth based and automatic re-initialization tracking results. The ground truth is annotated at every 15 frames. In the remainder of this section *error* refers to the 3D Euclidean distance between the algorithm output and the ground truth at an annotated frame. The sequence *mean error* (ME) is equal to the average error for all annotated frames. The average error produced by the tracking algorithm on all sequences is referred to as the *overall error* (OE).

We started the experiment using ground truth for initialization and drift detection mechanisms (*i.e.* compare the tracker output to the ground truth at every 15^{th} frame). We summarize these results in Table 1. The first two columns report the minimum and maximum overall error for various parameter choices of each tracker, respectively. The final column shows the average number of re-initializations per sequence for the two methods. This demonstrates that the performance of our proposed 3D tracker surpasses that of the baseline 3D tracker. What results is a 35% relative reduction in the minimum overall error, as well as a 70% relative reduction in average number of re-initializations. A more detailed version of these experiments can be found in [14].

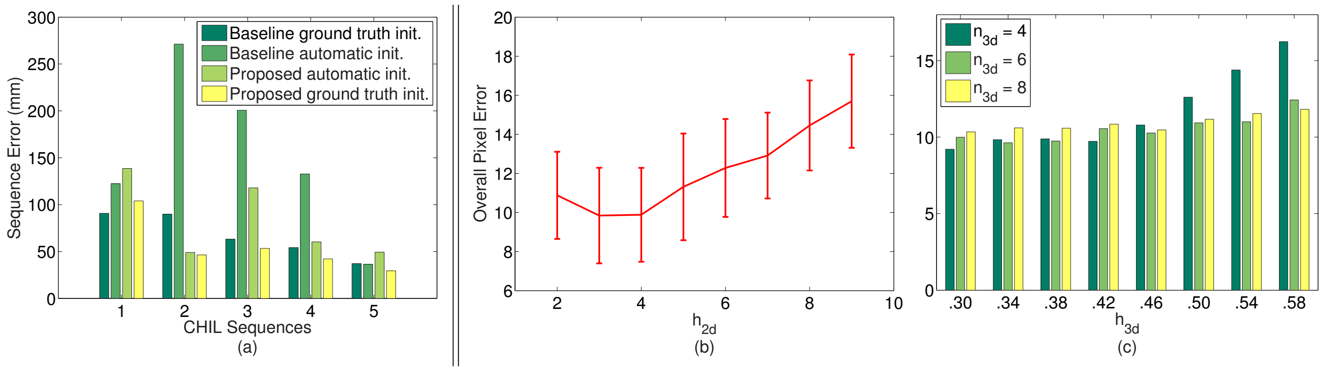

Figure 3. (a) Tracking results using ground truth re-initialization and our proposed automatic re-initialization on **CHIL dataset**. (b) Overall pixel error of the baseline tracker as a function of bandwidth h_{2d} and (c) the proposed tracker as a function of bandwidth h_{3d} and number of partitions n_{3d} for the **outdoor surveillance dataset** (with automatic re-initialization).

To automatically bootstrap the trackers, we first generated point clouds using the visual hull algorithm (Sect. 4.1) and clustered them using mean shift. For detecting drift in these indoor experiments we adopted the following distance metric. For a given cluster centroid $\mathbf{Z}^j$, we compute the Euclidean distance between the current tracker position and $\mathbf{Z}^j$. We normalize this distance by the maximum Euclidean distance from the set $\{\mathbf{Z}^j\}$ so that at a single iteration all distances are on the $[0, 1]$ scale. We denote this normalized distance as a. We also extract the color model for the centroid, $\hat{\mathbf{p}}(\mathbf{Z}^j)$ and compute the distance metric $b = \sqrt{1 - \hat{\rho}\left[\hat{\mathbf{p}}(\mathbf{Z}^j), \hat{\mathbf{q}}_{init}\right]}$ where $\hat{\rho}$ is the sample estimate of the Bhattacharyya coefficient between this distribution and the original color model $\hat{\mathbf{q}}_{init}$. We combine the two into a final distance $d = .5 * (a + b^2)$. The appearance information can significantly improve results in this case as we are only interested in the speaker's head, and clustering humans in the seminar room may be noisy and result in an over-segmentation. The original color distribution $\hat{\mathbf{q}}_{init}$ will help correct model drift and rediscover the head location allowing the tracker to recover from failure.

We present tracking results with automatic re-initialization on some sequences from the CHIL dataset in Fig. 3(a). In this bar graph we plot the sequence error of the baseline tracker and the proposed tracker using both ground truth and automatic re-initialization for these sequences. From these plots we see that in three of the five sequences, the proposed tracker outperforms the baseline tracker. In sequence one this is not the case. However, the baseline ground truth tracker also outperforms the proposed ground truth tracker, implying that the automatic re-initialization is consistent with the ground truth. And finally, in sequence five, the baseline tracker is able to outperform the proposed tracker slightly, but it is important to note that even in cases when the proposed tracker is outperformed with this re-initialization, it still compares to the ground truth re-initialization error.

Although the CHIL dataset provides a large number of annotated frames as well as fully calibrated cameras, it has the disadvantage of not providing a background model for each sequence of the dataset. This problem has recently been acknowledged by the CHIL consortium, and they plan to release background models for forthcoming datasets. As we began our experiments on this dataset, we quickly realized that we would not be able to create a reasonable background model for all sequences even using a state of the art method. We attempted to use codebook based background subtraction [9], but had little success due to the nature of the scenario (the main target we wish to extract using background subtraction, the speaker, moves so little that he/she assuredly becomes part of the background). We were only able to generate a reasonable background model for a few sequences some of which are reported here. This limits us from reporting the overall error due to automatic re-initialization on the entire dataset.

Therefore, to demonstrate the effectiveness of our algorithm on more diverse data and to overcome the limitations imposed by the absence of a background model in the CHIL database, we collected a new dataset on which we report results in the next section.

5.2. Outdoor Surveillance Database

To conduct this set of experiments, we recorded 27 sequences containing tracks of people walking in a busy area monitored by three synchronized cameras (Fig. 2, bottom row). Each camera recorded 320×240 pixel color images at 30Hz. The track lengths varied between 300–600 frames each, and the ground truth is manually annotated at every 15 frames. The goal of this experiment is to track people in the scene, and compare the error to the known ground truth. The dataset contains several natural instances of crowding, occlusions, view changes, etc. The typical person size in these images varied between 10×20 to 20×40 pixels. Due to the low resolution targets even a small 3–4 pixel error in manual ground truth annotation is significant.

We employed the auto-calibration approach of [13] to obtain a metric calibration space and derive the camera matrices $\mathbf{P}^i$. Since the calibration is accurate only up to a scale factor, we cannot report results in traditional units like millimeters as in the previous section. We therefore report results in pixel error, *i.e.*, sum of 2D projection error between the tracker output and the ground truth in each view.

We use the epipolar matching technique of Sect. 4.2 to create point clouds for the outdoor scenario. In this case the distance metric for cluster proximity (Sect. 4) was simply chosen to be d, the Euclidian distance between the current tracker location and the cluster centroids $\{\mathbf{Z}^j\}$. In the outdoor scenario the point clouds for each target (person) are often correctly segmented and hence a complicated appearance based distance metric is unwarranted.

We evaluated the effect of various algorithmic parameters on the performance of the two trackers on this dataset. Both trackers were automatically bootstrapped. First we address the critical parameter in 2D mean shift tracking, the bandwidth h_{2d}. Ideally this parameter should be optimally adapted over time, but as discussed previously, there exists no optimal method to do that. To simplify the comparison, we use the same bandwidth across all views, and analyze the effect of this parameter on the accuracy of the baseline tracking algorithm using our automatic re-initialization routine to detect drift. Results are shown in Fig. 3(b). As seen in the plot, the overall error for the baseline tracker varies between 9.8–15.7 pixels (error bars denote the standard deviation) on all sequences in the outdoor dataset. As expected, the algorithm exhibits large errors for smaller and larger bandwidth values, reinforcing the criticality of h_{2d}.

Further, we wished to see how the bandwidth parameter h_{3d} and the sampling parameter s_{3d} affected performance of our proposed feature level fusion 3D tracker. For a given sampling rate s_{3d} in each dimension, we can divide a cube of side $2h_{3d}$ in $(n_{3d})^3$ partitions where $n_{3d} = 2h_{3d}/s_{3d}$. We experimented with many different values of these parameters for the outdoor dataset. The bandwidth was varied from .25 to .6 and n_{3d} took on values 4, 6, 8. The results of the experiments are reported in Fig. 3(c). This graph, as one would expect, demonstrates that larger bandwidths require a higher number of samples to produce better results while smaller bandwidths perform better with a lower number of samples. The optimal performance is obtained at $h_{3d} \cong 0.3$ which is roughly the half-width of an average human in our calibration scale. Again, once we select these two parameters h_{3d} and n_{3d} for a given scene, they are fixed for the duration of tracking.

Another important factor for automatic drift detection and re-initialization is the frequency at which the tracker performance should be re-evaluated. Fig. 4(a) shows the overall pixel errors for the entire dataset for various values of parameter f_d, where drift detection is performed at every f_d-th frame. The graph exhibits an interesting pattern where both trackers perform well for an intermittent value of f_d and the overall error is higher for small and large values of the parameter. This behavior is explained by the fact that if we rely on the point-cloud clustering method for re-initialization very frequently (*e.g.*, at each frame) then the clustering errors will propagate into final results. On the other hand, if we wait too long to evaluate the tracker performance then it might be too late for the tracker to recover in case of mistakes/drifts. Also note that proposed tracker is again consistently superior in performance as compared to the baseline tracker as seen in the figure.

Fig. 4(b) compares the sequence pixel errors for the 27 sequences for both the baseline and the proposed trackers. The feature fusion-based 3D tracker outperforms the decision fusion-based 3D tracker in most cases. Also, in Fig. 4(c) we show representative tracking results (per frame tracking errors) on a particular sequence. Again the proposed tracker has lower error than the baseline tracker for the entire sequence.

Finally, we would like to compare the best overall results produced by each method against the ground truth re-initialization results. The baseline tracker produced the lowest overall tracking error of 9.84 pixels for $h_{2d} = 3$ and $f_d = 15$ whereas the lowest overall error of 7.99 pixels was obtained for the proposed tracker for $h_{3d} = 0.315$, $n_{3d} = 4$, and $f_d = 4$. In other words, a 19% relative reduction in overall error can be obtained by using the proposed tracking approach over the baseline method. The ground truth based initialization errors for the proposed and the baseline methods were 6.19 and 6.76 pixels, respectively.

5.3. Discussion of Results

The experiments carried out on the CHIL dataset (Sect. 5.1) using the ground truth based initialization clearly show that the proposed feature fusion tracker clearly outperforms the decision fusion baseline tracker (Table 1). Additionally, the point cloud based automatic initialization and drift detection results are mostly in agreement on the tracker performance (baseline vs. proposed) and only deteriorate (as expected) slightly compared to the ground truth results.

The tracking results from the experiments conducted on the outdoor surveillance dataset (Sect. 5.2), firstly, demonstrate that our algorithm is applicable in various domains (*e.g.* it can successfully track people at low resolutions). Secondly, throughout the dataset it is evident that our proposed tracker consistently outperforms the baseline tracker, supporting our findings from Sect. 5.1. Finally, the automatic re-initialization scheme for the feature-fusion based tracker on average is only 1.8 pixels from the tracking error using ground truth re-initialization showing the effectiveness of the bootstrapping approach.

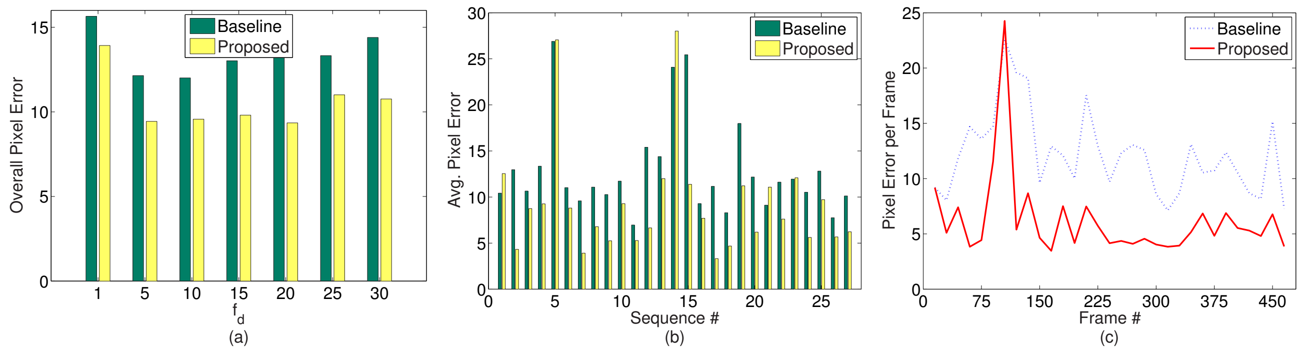

Figure 4. Results of feature fusion (3D) vs. decision fusion (2D) tracking approaches. (a) The overall pixel error for different values of f_d. (b) Pixel error for each sequence. (c) The pixel error per frame on one sequence.

6. Summary and Conclusion

We introduced a fully automatic kernel-based 3D tracker. The tracker fuses information from multiple cameras at the feature level as opposed to the decision level. This formulation allows the tracker to resolve the problems of scale selection, occlusion, view dependence, and correspondence across different views in an elegant manner. The tracker is bootstrapped with an automatic re-initialization technique based on clustering 3D point clouds of foreground objects.

The proposed automatic framework is evaluated experimentally on two datasets. Both datasets contained several instances of background clutter, distracters, occlusions, view changes, etc., that have been effectively dealt by presented tracker. Furthermore, these experiments conclusively demonstrate that the feature level fusion approach outperforms the decision level fusion approach to 3D tracking, obtaining a 35% and 19% relative reduction in error on the two datasets, respectively. We also show that our bootstrapping scheme performs comparably with ground truth re-initialization. The two datasets obtained from different scenarios exhibit the wide applicability of the algorithm.

We intend to extend this study for further investigation of sensor fusion techniques for the purpose of automatic video monitoring and surveillance.

7. Acknowledgements

This research was supported in part by the National Science Foundation under grant No. 0236653.

References

[1] The chil consortium web-site. `http://chil.server.de`.

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

[3] R. Collins. Mean-shift blob tracking through scale space. In *Proc. Comp. Vis. and Pattern Rec.*, pages 234–240, 2003.

[4] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Patt. Analy. and Mach. Intell.*, pages 603–619, may 2002.

[5] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Trans. Patt. Analy. and Mach. Intell.*, pages 564–577, may 2003.

[6] T. Darrell et al. Plan-view trajectory estimation with dense stereo background models. In *Proc. Int. Conf. Comp. Vis.*, pages 628–635, 2001.

[7] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[8] M. Isard and J. MacCormick. BraMBLe: A Bayesian multiple-blob tracker. In *Proc. Int. Conf. Comp. Vis.*, pages 34–41, 2001.

[9] K. Kim et al. Real-time foreground-background segmentation using codebook model. *Real-time Imaging*, 11(3):172–185, June 2005.

[10] J. Krumm et al. Multi-camera multi-person tracking for EasyLiving. In *Proc. Work. Vis. Surveillance*, 2000.

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

[12] 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.

[13] M. Pollefeys et al. Visual modeling with a hand-held camera. *Int. J. of Comp. Vis.*, 59(3):207–232, 2004.

[14] 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 1–1, 2007.

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

[16] Z. Zhang et al. A joint system for person tracking and face detection. In *Proc. IEEE Int. Work. Human Comp. Interaction*, Beijing, China, 2005.

[17] Q. Zhou and J. K. Aggarwal. Object tracking in an outdoor environment using fusion of features and cameras. *Image and Vision Comp.*, 24(11):1244–1255, 2006.