Appears in: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 697-703, June 1997.

Real-Time Closed-World Tracking

Stephen S. Intille, James W. Davis, and Aaron F. Bobick
MIT Media Laboratory
20 Ames Street – Cambridge – MA 02139 – USA
intille | jdavis | bobick @media.mit.edu

Abstract

A real-time tracking algorithm that uses contextual information is described. The method is capable of simultaneously tracking multiple, non-rigid objects when erratic movement and object collisions are common. A closed-world assumption is used to adaptively select and weight image features used for correspondence. Results of algorithm testing and the limitations of the method are discussed. The algorithm has been used to track children in an interactive, narrative playspace.

1 Introduction

Many video understanding tasks require observation and tracking of multiple rigid and non-rigid objects undergoing rapid, unpredictable motion. Consider the following scenario:

> Four six-year-old children begin the game in the "bedroom," an 18 by 24 foot space furnished like a child's bedroom, complete with a movable bed. As the children frantically run around the room, objects like rugs and furniture begin to "speak" based on the children's actions. Suddenly, the room transforms — images of a forest fade onto two of the room's walls. The children embark upon an interactive adventure, while being instructed and observed by the room.

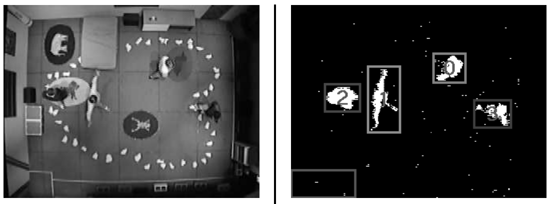
We have constructed such an imagination space, called the KIDSROOM, where a computer system tracks and analyzes the actions and interactions of people and objects[4]. Figure 1-a shows the experimental setup of the KidsRoom viewed from the stationary color camera used for tracking objects. Here four people are in the scene. During a typical multi-person interaction, particularly when the people are young children, similar-looking objects bump into one another, move by one another, and make rapid, large body motions.

In this paper we describe a real-time tracking algorithm that uses *contextual information* to simultaneously track multiple, complex, non-rigid objects. By context we mean knowledge about the objects being tracked and their current relationships to one another. This information is used to adaptively weight the image features used for correspondence. The algorithm was designed for use in the KidsRoom playspace. The goal of this paper is to detail generic problems and solutions that are in any similarly complex tracking tasks.

2 Previous approaches

Most real-time tracking systems that track non-rigid objects have been designed for scenes containing single, large objects such as a person[17], hand[2] or face[6] and have not been tested in domains where independent, multiple objects can interact.

Adaptive correlation template tracking methods (e.g. energy-based deformable models[2]) typically assume each template varies slowly or smoothly, templates don't collide, and high-resolution support from the data is available. As we will discuss, we have found it is often difficult to separate the tracking from the boundary estimation problem. Therefore, some people-tracking methods that require accurate object boundary detection based on intensity[1] and optical flow[12] are difficult to apply in our tracking task. Further the difficulty in segmenting colliding children and their rapid changes in appearance and motion prevents the use of differential motion estimators that use smooth or planar motion models [3] and tracking techniques that require reasonably stable edge computation[7]. Motion difference blob size and shape characteristics are sometimes used for



(a) (b)

Figure 1. (a) The top-down view of the KidsRoom used by the real-time tracking algorithm. Four people and a movable bed are in the room. (b) By using a fast, simple clustering algorithm, small background difference clusters are merged with larger nearby clusters, as marked by the bounding boxes.

tracking[15]; here we use background-differenced blobs. However, since children are small and their difference blobs merge frequently, there is usually no single feature for an object that remains trackable for more than a few seconds.

Consequently, we believe that low-resolution, non-rigid, multiple-object tracking in real-time domains like the KidsRoom requires using contextual information to change how different features are used for tracking based on contextual information. Systems that have used non-geometric context in dynamic situations include Fu's shopper system[5], Prokopowicz' active vision tracker[13], and Rosin's context-based tracking and recognition surveillance system[15]. Here we modify a tracking approach that uses context-sensitive correlation templates for tracking non-rigid objects[8] for a real-time application.

3 Closed-worlds

Strat[16] has demonstrated that context-dependent visual routines are powerful tools for image understanding in complex static domains. *Context* is one way of addressing the knowledge-selection problem in dynamic, multi-object tracking. We consider the context of a tracking problem to be a boundary in the space of knowledge — a boundary outside of which knowledge is not helpful in solving the tracking problem[8]. In the KidsRoom domain, a context could be

> "a region of the room away from the door that contains two children and the floor; one child has a "reddish" average color and was last moving left; the other child has a "bluish" average color and was last moving right."

One way to generate and exploit such contextual knowledge is to use a *closed-world* assumption. A closed-world is a region of space and time in which the specific context of what is in the region is assumed to be known. The internal state of the closed-world — e.g. the positions of the objects — however, is not necessarily known. Visual routines for tracking can be selected differently based on knowledge of which other objects are in the closed-world. Closed-worlds circumscribe the knowledge relevant to tracking at a given instant and therefore reduce the complexity of the tracking problem. Nagel[11] and Mundy[10] have both suggested that closed-world information is useful for building systems that extract conceptual descriptions from images and image sequences.

For robust tracking in a complex scene, a tracker should understand the context of the current situation well enough to know which visual *features* of an object can be tracked from frame to frame and which features cannot. Based on these observations, a technique is described in [8] called closed-world tracking. In that off-line implementation, adaptive correlation templates are used to track small, non-rigid, colliding objects — players on a football field. Closed-world contextual information is used to select pixel features for inclusion in the matching template; the method for selecting the pixels changes based upon which objects are in a given closed-world.

The remaining sections of this paper describe a real-time implementation of closed-world tracking. Closed-world information is used to adaptively weight different types of image features into a single correspondence measure based on the current context. The closed-world assumption is also used to determine the order in which different matching situations should be considered.[1]

The details of our real-time closed-world tracking algorithm differ from those described in [8] because of the limitations introduced by real-time requirements and differences in the imaging environment. However, the nature of the two tracking systems is similar. In the following sections we describe the algorithm in detail and some problems we have encountered; many of the difficulties are generic and will need to be addressed by anyone with similar tracking tasks.

4 Objects, blobs, and closed-worlds

The tracking algorithm uses four data structures: (1) Each *object* in the world has a data structure that stores the object's estimated size, color, velocity, and current and past position. This information is used for matching each object in the last frame to a blob in the new frame. (2) Image *blobs* are computed in each frame using background differencing; each blob's size, color, and position is recorded. (3) A *local closed-world* data structure exists for every blob, as in [8], and stores which objects are assigned to the blob and how long they have been there. Two objects that are touching

[1] Mundy[10] has suggested that given a closed-world a system should assume a simple explanation first and then if that explanation is not consistent with the data consider more complicated explanations.

or nearly touching will appear as one blob; therefore, they should be assigned to the same closed-world. The state of the closed-world to which an object is assigned determines how the object's properties are re-estimated from the current frame. (4) Finally, the system uses knowledge about the the *global closed-world*, which stores information about which objects are in the entire scene.

At each time step, the algorithm must match objects in the last frame to blobs in the new frame using the object and blob properties such as color and position. The metric used for matching is a weighted sum of several individual feature distance measures, where the local and global closed-world information from the last frame is used to set the weights and to control the matching strategy. Once all objects are matched to blobs, the object properties are updated using the new blob information and the state of the new closed-world to which it has been assigned. The algorithm uses global closed-world information to determine if any objects entered or left the scene. The algorithm then iterates. The following sections describe each stage in more detail.

5 Computing blobs

Recall that the room is configured as shown in Figure 1-a. The camera is static, and therefore a reliable background image free of people and movable objects can be obtained.

Due to the difficulty of modeling rapidly moving people (particularly small children), background-difference blobs are used as the primary visual representation. The background is removed from each frame using a YUV-based background subtraction method similar to the method described in [17], which uses color but not intensity intensity information. This method correctly removes most shadowed regions.

In practice the resulting thresholded blob images are noisy due to camera noise or objects that are broken apart because they have regions colored like the background. Therefore, three dilation operations are performed and then a fast bounding box merging algorithm is used to cluster small groups of blobs.[2] The resulting blob image for one frame, with bounding boxes drawn around clusters, is as shown in Figure 1-b.

Each blob's size, position, and average color are computed during the clustering operation, but using the original, *not the dilated*, threshold image. Using the dilated image will corrupt the size and color estimates because the blob will contain many background pixels. Unfortunately, the non-dilated image is sensitive to the color thresholds used by the background differencing algorithm. The threshold must be balanced at a setting that can discriminate most clothing from the background while not including many background pixels in the difference blob.

6 Matching objects to blobs

This section describes the image features used for matching, how those features are combined into a single match score matrix, and how the match score matrix is used to determine object-to-blob correspondence.

6.1 Real-time features

Four properties are computed by each object — average color, position, velocity, and size — and used to compute matching distance measures.[3] Due to blob thresholding errors, each of these estimates can be noisy as a person moves around the room.

The first measure is distance between the average color of an object and a blob. Average color is a reliable discriminator between many objects when (1) shadow pixels are a small percentage of blob pixels, (2) blob color is re-estimated as often as possible, and (3) the color is normalized for brightness. When the frame rate is about 3 Hz or higher, average color change between two frames for the same object blob is small compared to the color difference between most objects.

The second measure is the Euclidean distance between an object's position and a blob position in the new frame. At high frame rates, objects are normally close to their blob in the new frame.

The third measure is distance from predicted position. Velocity is estimated using an object's current and previous positions. Then the position of the object is predicted in the new frame and the Euclidean distance computed between the predicted position and the blob position in the new frame.

Finally, the fourth measure is the size difference between an object's current blob and all blobs in the new frame, which vary slowly at high frame rates.

6.2 Computing the match score matrix

For each distance measure, a match score matrix S is created, where entry S_{ij} is the match score of object i with

[2] The thresholded, dilated image is received after background removal. Connected regions are found, and their bounding boxes are computed. The distance between a cluster and and surrounding clusters (based on the closest bounding box points) is determined. If two clusters are within a small distance of one another (5 pixels in our system), the clusters are merged into a single cluster. No morphological growing is used to connect the merged pixels, however. They are simply assigned to the same cluster. A chaining process is then invoked where the current cluster continues growing until all nearby clusters have been considered for inclusion based on the bounding boxes of the clusters already included. The process repeats until all clusters are merged or found to be too far from other clusters to merge. Large diagonally-oriented objects have large bounding boxes. Hence, when merging a large cluster the bounding box difference is replaced with a slower algorithm that estimates the smallest distance between the boundaries of the two blobs.

[3] These four properties have been selected because they are quickly computable. An off-line or future system, however, could augment this information with histogram-based color algorithms and/or image templates as used in [8].

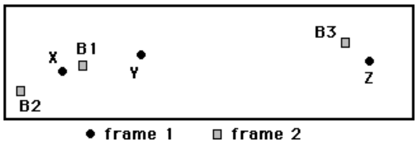


Figure 2. Sometimes the match that is the best locally is not the best when all other matches within a closed region are considered. When distance traveled is used as the feature, object X to blob B1 is the best match for X but creates a poor match of Y to B2.

blob j. To simplify the combination of measures, each S matrix is normalized such that $\sum S_{ij} = 1$.

Next, the four S matrices are weighted and summed into a single matching score matrix, M. M_{ij} is the single distance measure between object i in the last frame and blob j in the new frame. As discussed shortly, the weighting depends upon the local and global closed world situation, such as whether objects are merging, splitting, entering the room, or exiting the room. The matching has several phases and different weights are used for each step.

6.3 Computing the best match

Given M, the global closed-world assumption can be used to make object to blob matches using information about all known objects simultaneously. Consider the situation where the only active matching measure is closest distance. Figure 2 illustrates a scenario where the total number of objects in the world is known to be three. Objects X, Y, and Z in frame 1 need to be matched to blob objects B1, B2, and B3 in frame 2. Considering only the distance measure, Z clearly matches to B3. If matching independently, the best match for X is B1. If this match is made, however, the distance between Y and B2 (the only remaining match possible) is large. The best match for X is actually B2 because it keeps the distance between Y and it's best match, B1 low while still providing a good match for X.

Rangarajan and Shah[14] have described a non-iterative greedy algorithm that, given a score matrix with m objects, can be used to make object-to-blob matches in $O(m^2)$.[4] The algorithm will find a near-optimal pairing of all objects, avoiding individual assignments that are bad while trying to keep each match cost as small as possible. The reader is referred to their paper for the details of the algorithm. Since the tracking algorithm always knows the number of objects in the scene, we use the Rangarajan and Shah algorithm to compute correspondence given the match score matrix, M. We discuss a problem with this approach in section 8.3.

[4]Given we test our system on only four objects, we could compute the best global match for little additional cost, but for many other domains with more objects the greedy algorithm would be required.

6.4 Enforcing hard constraints

Hard constraints can be used to prohibit a match between object i and blob j by flagging M_{ij} as invalid. Two constraints currently used by the system are a maximum distance constraint and a size explanation constraint. The closed-world context controls which constraints are applied at a given stage in the matching process.

The maximum distance constraint eliminates the possibility of any match between an object and a new blob that is greater than some reasonable maximum distance. In Figure 2, therefore, object Z is so far from blobs B1 and B2 that the match is not even considered. In our system the maximum distance is set at 100 pixels, or about a third of the scene.[5]

A second hard constraint can prohibit object-to-blob matches that are inconsistent with size information known about the objects already assigned to the blob. When an object of size s pixels is matched to a blob, s pixels are marked as "assigned." If the correspondence algorithm later proposes that another object match with the same blob, there must be enough remaining unassigned blob pixels to accommodate the second object. [6]

7 Closed-worlds in closed rooms

Closed-worlds are used in three ways. (1) The global closed-world is used to determine the order in which matching should proceed in order to correctly maintain the number of objects in a "closed room" as objects enter and exit. (2) Knowledge about which objects are in local closed-worlds and the global closed-world is used to select the combination weightings used to generate the match score matrix M and to determine which matching constraints should be applied. (3) After matching, an object's local closed-world is used to determine whether some object attributes (e.g. size, velocity) should be re-estimated or held constant when the object is matched to a new blob.

7.1 The global closed-world

The global closed-world assumption requires that the tracking algorithm have an accurate count of the number of objects in the entire scene. In this domain, however, as in many "room" domains, initialization can be performed automatically by emptying the space of all movable objects and then defining a "closed-room" the same way that one is obtained in real life: by adding a "door." The one rule for

[5]This constraint is set rather high because when an object collides with other objects, each object's position is estimated from the center of the group. Hence, large changes in estimated position are possible.

[6]In the current implementation, "enough" is half of the pixel size of the object being added. Inaccuracies in size estimation are caused by shadows that are not fully removed, camera noise, and objects that look similar to the background.

the space is that all people and objects must enter one at a time through the door.[7]

The door is represented by a small region of the image, marked by the box in the bottom left corner of Figure 1-b. Objects are permitted to enter and exit the room through the door once the tracker is initialized. Thereafter, the tracker will maintain the global closed-world. The correspondence algorithm described later in section 7.2 makes use of the fact that objects don't appear out of thin air; hence, the order of matching can be used to reduce the chance of matching error.

7.2 Match order and feature weighting

We use a staged matching process that is ordered so as to use global closed-world information to minimize the chance of incorrect matches to due objects that are entering and leaving the room. The main algorithm can be broken into nine stages.[8] The weightings used to create the context-sensitive matching measure matrix M are set differently depending upon the matching step. For example, the second step is matching any known room object to any blobs already assigned a blob in the first matching step – any unmatched room objects are tested against occupied blobs that are big enough to contain more than one object. In this context color is not used as a matching feature, since it is known that the color average will be invalid when comparing an object to a blob that contains multiple objects.

[7]The algorithm is currently designed so that only one person is in the "door" at a time, which simplifies the vision. However, one can imagine placing additional cameras in the room that focus on the door and perform special processing to recognize who and what comes through. Our current system cannot correctly interpret a person walking through the door with an object and later dropping that object unless the system is explicitly told the new person has an object.

[8]Space prevents us from including the details of how weights are adjusted during each step; readers are referred to the extended version of this paper[9]. Briefly, the matching stages are as follows. *Step 1: Known object to unoccupied blob.* First, match objects that are known to be in the room in the last frame to *unoccupied* blobs in the new frame. Blobs in the door are temporarily ignored. Perform the most reliable matching first. *Step 2: Room object to occupied blob. Step 3: Door object to door blob.* Since only one person or object is allowed in the door at a time, if there is one unassigned object in the door and one blob in the door, assign the object to the door blob. *Step 4: Room object with door blob.* If a door blob is not explained by assignment of an object in the door, then try to explain it by matching it with an object remaining in the room that did not match to any room blobs. *Step 5: Door object with room blob.* This is the case where an object was in the door one frame then moved into the room the next but where the object existed previously in the scene and did not enter through the door. *Step 6: Door blob without object; make object.* If all objects are assigned to blobs by this step and a door blob exists, the only explanation is that a new object just came into the door. A new object is created. *Step 7: Door object without door blob; remove object.* By this stage, if an object exists in the door it has no blob match in the door or the room. Therefore, the object must have left the room through the door; the object is removed. *Step 8: Last-resort.* At this point, the algorithm should have been able to explain every situation. If not, the system can try to automatically handle the error by loosening some constraints (e.g. size) and then using a "best guess" strategy. *Step 9: Failure.* Finally, if all objects are assigned to blobs and there are blobs without objects assigned to them left over, the algorithm has no "best guess" option and can simply signal an error.

The stages are ordered to exploit the fact that objects can't appear from or disappear into "thin air" and work from the most common to the least common explanations.

7.3 Updating object properties

After object assignment is completed, the local closed-world is used to determine whether an object's properties can be reliably updated from it's blob given a a particular context.

Color estimation only occurs when an object is the only object in its closed-world. During these times, the average color will not be corrupted by other objects. Hence, when two objects are assigned to the same closed-world, each object's average color is held constant. Color estimation resumes only when an object is isolated again in a single-object closed-world.

The same strategy controls size estimation. When objects are in single object closed-worlds, the object size is re-estimated from the current blob, but when an object is in a multi-object closed-world, its size tag is held constant.

An object's position is updated every frame regardless of closed-world status using the centroid of the object's blob. Therefore, two objects in the same closed-world a given frame will have the same position. Consequently, when the objects split into separate closed-worlds the distance measure will not affect the matching since both objects will be the same distance from all candidate blobs.

Velocity estimates will be error-prone when objects move from multi-object closed-worlds to single-object closed-worlds and visa versa.[9] Hence, the velocity is not re-estimated when objects are in multi-object closed-worlds. Therefore, when matching objects in a multi-object closed-world that have just split into multiple single closed-worlds, the velocity measure will prefer matches that have objects moving in the same direction as they were moving when they originally merged. This preference is desirable if the objects are merged a short time (probably just passing nearby each other), but the longer the objects are together, the more likely it becomes that one or both objects have changed velocity. Therefore, an additional parameter is stored for each object, the merged-time, or the amount of time the object has been in a multi-object closed-world. The weight of the velocity feature is scaled by the a merged-time-weight value which falls off gradually from 1.0 for a few frames before being thresholded to 0 after about 1 second.[10]

[9]The position of each object in a multi-object closed-world is the centroid of the closed-world, which can be far enough from the actual position to create a spurious velocity measurement when each object returns to a single closed-world and the blob position suddenly shifts to the center of the single object.

[10]The merged-time-weight is set as exp(-sqr(MERGED-TIME)/16) with a cutoff to zero after 4 frames.

8 Tracking results

The real-time tracking system takes s-video input from a color Sony HandiCam. Images, 320 by 240 pixels, are processed at approximately 7 Hz on a single SGI Indy 180MHZ R5000 processor when one adult is in the room and 5 Hz when four adults are in the room.[11] Debugging and testing was performed by recording multiple people on VHS tape and playing the recorded data back into the real-time tracking system to classify the interactions and study errors.

8.1 Typical situations

To evaluate the algorithm we have recorded about 12 minutes of test video, with up to four people in the space at once. Four of the people, as seen by the camera, are shown in Figure 1-a. Some of the video contains people wearing brightly-colored rain ponchos. The people were asked to walk or jog in the tracking space and to perform several different types of movement that could fool the tracker: *isolated movement*, *entry and exits*, *pass-bys*, *in-and-outs*, *merge-and-stops*, *merge-and-travels*, *large group merges*, and *illegal activity*. [12]

Since the algorithm is running at 5-7 Hz, performance can sometimes change from run to run on a single sequence depending upon the frames that happen to be used. Therefore, each test sequence was evaluated twice. Table 1 shows the number of each type of situation we observed for poncho and regular data in our test data and the number of times our tracker mislabeled any tracked objects for that type of situation.

8.2 Performance

The algorithm's best performance is, predictably, on single-object tracking and collisions of two objects. In these cases the method's adaptive feature weighting strategy usually leads to a correct match even when two objects have similar-looking clothing (e.g. two different shades of blue) or are making large body motions and moving rapidly around the room. The algorithm can reliably maintain the number of objects in the space as objects enter and exit, and it can be tuned so that it almost never "loses" an object, even though it may swap two object labels (as tested extensively on children in the KidsRoom[4] system).

Not surprisingly, our algorithm generally performs better when the objects being tracked have distinctive color features, like colored ponchos. However, when color features are weak *and* when three and four objects merge into a large group, matching performance suffers.

8.3 Limitations

Analysis of the system's matching errors revealed five unanticipated limitations of the real-time closed-world architecture that are issues common to many generic tracking tasks.

First, the system is overly-reliant on blob data. There are two main problems with the blob representation: (1) regardless of how the color difference thresholds are set, the blobs will not perfectly segment objects; the errors in blob computation then propagate to every feature used for matching, and (2) multi-object blobs provide little meaningful information that can be used for matching and re-estimation of object properties. A solution when more processing power is available may be to use other features that can be computed independently from the blob regions, such as correlation templates, and to partition merged blobs where possible (see [8]).

A second, related architectural limitation is that the algorithm has no mechanism for handling slow variation of image features while objects are merged in a large closed-world. The algorithm is designed to adapt each frame to slowly varying matching features. During a merge, however, when an object is in a single-object closed-world, color, velocity, and size measurements are not updated. Consequently, when objects merge and travel, matching errors are more likely because objects change appearance (e.g. see *merge and travels* in Table 1). Even if objects merge and don't travel, one object can move limbs and change significantly in size and corrupt subsequent matching.

The third architectural issue is match evaluation. The algorithm uses the Rangarajan and Shah greedy correspondence algorithm[14] to avoid bad global matches when making individual matches, but this strategy leads to some bad matches (e.g. most of the *pass-by* errors). For instance, whether the tracker makes the correct, best scoring match of object o_1 to blob b_1 depends upon how it matches object o_2. The problem is that o_2 may have just been in a multi-object blob or may be matching to a multi-object blob, and all possible matches may have mediocre scores. o_1's match, however, can change depending on whether $matchscore(o_2, b_3)$ or $matchscore(o_2, b_2)$ has the lowest value – even when both match scores are poor and the small difference between them provides no useful information. Consequently, even when match o_1 to b_1 has the best match score for *every*

[11] There is no limit to the number of people that can be tracked, but performance suffers as the number of pixels that must be clustered increases.

[12] *Isolated movement* is the most basic tracking situation, when one individual is moving around the space isolated from all other individuals. *Entry and exits* are when a person enters or leaves the space via the door. *Pass-bys* are when individuals move near one another so that they are close enough to merge as they walk or run past each other. *In-and-outs* are a special test action where two or more individuals stand next to one another so they are apart and rapidly lean in and out so they merge and split again. *Merge-and-stops* are when two or more people walk towards each other to merge and then stop for several seconds before splitting again. *Merge-and-travels* are when two or more people merge, then walk around the room very close together before they split again. *Large group merges* occur when three or more people merging together and splitting. Finally, *illegal activity* is when a person enters or leaves the room space without going through the door or when someone drops an unknown object and walks away.

	isolated movement	entry and exits	pass-bys	in and outs	merge and stops	merge and travels	large group merges	illegal activity
Examples Poncho	51	14	17	5	6	0	10	0
Poncho errors (run 1)	0	0	0	0	0	0	1	0
Poncho errors (run 2)	1	0	0	0	0	0	3	0
Examples Regular	133	27	34	15	16	7	10	3
Regular errors (run 1)	0	0	3	0	1	2	6	0
Regular errors (run 2)	0	0	4	0	1	4	7	0

Table 1. This table shows the number of each type of tracking situation that appears in our test sequence with up to four people being tracked at once. The events were classified manually. For each test, the number of times the tracker made an error is indicated.

matching measure (i.e. distance, color, size, and velocity) and all other matches are about equally poor, o_1 may match incorrectly. In short, bad isn't always terrible, and the algorithm needs non-linear, context-sensitive cutoffs that distinguish between the two. When the scene contains several merged objects, the chance of such an error will increase because three out of four objects may have mediocre (but not terrible) match possibilities that lead to the algorithm ignoring the best match.

Fourth, object properties are multi-stable because they depend upon object assignment to blobs. For example, if objects o_1 and o_2 are assigned to blob $b1$, they both have position P_1. However, if o_3's blob then merges, the position of all three objects will be P_2, where P_1 and P_2 can be over 20 pixels apart. Similarly, color is unstable depending upon the number of objects in a closed-world.

Finally, the fifth architectural problem is that the system is forced to make a binding matching decision at every frame. There is no mechanism for detecting a questionable match and then looking for further match evidence over 5 or 10 frames before making a final decision. When the different match possibilities have similar evaluation scores, noise and unusual merge configurations can lead to spurious matches that might be corrected if several frames were considered.

Because of these architectural limitations, the system is very sensitive to some thresholds (e.g. the size constraint and background differencing thresholds) in situations like multi-object merges. Unfortunately, every threshold setting has its problems. Significantly improving the algorithm probably requires changing the architecture to address the problems described above.

9 Summary

In this work we have used closed-world regions to perform context-based tracking in a real-t ime domain where object motions are not smooth, small, or rigid and when multiple objects are interacting. The tracking system has been used for the KidsRoom[4], an interactive, narrative children's playspace, and could prove useful for other overhead tracking domains like observation of city street intersections, sporting events, pedestrian malls and parking lots.

References

[1] A. Baumberg and D. Hogg. Learning flexible models from image sequences. In *Proc. European Conf. Comp. Vis.*, volume 1, pages 299–308, Stockholm, Sweden, May 1994.

[2] A. Blake, R. Curwen, and A. Zisserman. Affine-invariant contour tracking with automatic control of spatiotemporal scale. In *Proc. Int. Conf. Comp. Vis.*, pages 66–75, Berlin, Germany, May 1993.

[3] S.D. Blostein and T.S. Huang. Detecting small, moving objects in image sequences using sequential hypothesis testing. *IEEE Trans. Signal Proc.*, 39(7):1611–1629, 1991.

[4] A. Bobick, S. Intille, J. Davis, F. Baird, L. Campbell, Y. Ivanov, C. Pinhanez, A. Schutte, and A. Wilson. The KidsRoom: A perceptually-based interactive and immersive story environment. M.I.T. Media Laboratory Perceptual Computing Section 398, M.I.T. Media Laboratory Perceptual Computing Section, November 1996. Revised September 1997, see also http://vismod.www.media.mit.edu/vismod/demos/kidsroom.

[5] D.D. Fu, K.J. Hammond, and M.J. Swain. Vision and navigation in man-made environments: looking for syrup in all the right places. In *Proc. Work. Visual Behaviors*, pages 20–26, Seattle, June 1994.

[6] G.D. Hager and K. Toyama. The XVision system: A general-purpose substrate for portable real-time vision applications. *Comp. Vis., Graph., and Img. Proc.*, 1996. To appear.

[7] D.P. Huttenlocher, J.J. Noh, and W.J. Rucklidge. Tracking non-rigid objects in complex scenes. In *Proc. Int. Conf. Comp. Vis.*, pages 93–101, Berlin, Germany, May 1993.

[8] S.S. Intille and A.F. Bobick. Closed-world tracking. In *Proc. Int. Conf. Comp. Vis.*, pages 672–678, June 1995.

[9] S.S. Intille, J.D. Davis, and A.F. Bobick. Real-time closed-world tracking. MIT Media Lab Perceptual Computing Group Technical Report No. 403, MIT, Nov. 1996.

[10] J. Mundy. Draft document on MORSE. Technical report, General Electric Company Research and Development Center, Feb. 1994.

[11] H.-H. Nagel. From image sequences towards conceptual descriptions. *Image and Vision Comp.*, 6(2):59–74, 1988.

[12] R. Polana and R. Nelson. Low level recognition of human motion. In *Proc. Work. Non-Rigid Motion*, pages 77–82, Nov. 1994.

[13] P.N. Prokopowicz, M.J. Swain, and R.E. Kahn. Task and environment-sensitive tracking. In *Proc. Work. Visual Behaviors*, pages 73–78, Seattle, June 1994.

[14] K. Rangarajan and M. Shah. Establishing motion correspondence. *Comp. Vis., Graph., and Img. Proc.*, 54:56–73, 1991.

[15] P.L. Rosin and T. Ellis. Detecting and classifying intruders in image sequences. In *Proc. British Mach. Vis. Conf.*, pages 24–26, Sep. 1991.

[16] T.M. Strat and M.A. Fischler. Context-based vision: recognizing objects using information from both 2D and 3D imagery. *IEEE Trans. Patt. Analy. and Mach. Intell.*, 13(10):1050–1065, 1991.

[17] Christopher Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. Patt. Analy. and Mach. Intell.*, 1997.