View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DSpace@MIT

Multi-Person Tracking Using Dynamic Programming

by

Rania Y. Khalaf
B.S., Massachusetts Institute of Technology, 2000

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2001

© Massachusetts Institute of Technology 2001. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by .....
Trevor Darrell
Assistant Professor of EECS
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Multi-Person Tracking Using Dynamic Programming

by

Rania Y. Khalaf

B.S., Massachusetts Institute of Technology, 2000

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2001, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The tracking algorithm presented in this work makes assignments of people to foreground blobs in each frame by choosing the assignments that are most strongly supported over a window of time. Real-world constraints such as color constancy and motion continuity are encoded using inter-frame histogram intersection and intra-frame distance measurements. These measurements are then used to connect a graph of hypotheses at each frame. The graph is searched using dynamic programming for the best decision based on the least cost path from the frame about to be committed till the newest frame taken. Contextual information about how close people are to each other and how long they have been apart is used to decide which features to update when an assignment is made. The algorithm was implemented using a static overhead Intel color camera in an indoor space on a 300MHz dual-Pentium II PC using 160x120 resolution images. Reliable results were obtained with 30 minutes of video of two to four people interacting in the space. People walked around, met and talked, lay down on the bed, or sat down on the chairs or floor.

Thesis Supervisor: Trevor Darrell
Title: Assistant Professor of EECS

Acknowledgments

I would like to thank...

My family in Beirut for putting up with my phone calls that invariably woke them up, coming to visit, and giving me a good laugh when I was too stressed out to make any sense.

Stephen Intille, for his help with ideas and implementation possibilities through out and his support of my work.

Trevor Darrell for his guidance, his timely and direct responses when things were unclear, and the great chair in his office.

Patrick Henry Winston, my great academic advisor who helped me out in some really tough times at the institute, and from whom I have learned a great deal about everything and why it works. Thank you, Patrick for being a friend as well.

My friends. Jean Claude Saghbini for the long rania's-thesis-idea-bouncing sessions after his own rough days at work. Delphine Nain for being a live, instant, Java helpline. Hala Qanadilo and Jeremy Gerstle for the many days of panic and quick lunches. Everyone who walked around the room for my videos.

and House_n for sponsoring this work.

Contents

List of Figures

Chapter 1

Introduction

The goal of this work is to robustly track multiple people in an indoor, home environment from visual input. In such an environment, people sometimes group together. They also sometimes stay in the same location for extended periods of time to watch television, read, or rest. Kids may run around and play. Therefore, the algorithm needs to handle people of different ages and activity levels as they move around the space. The target number for our application was three to four people, which is around the 2.64 average number of people per household in the US in the 1990s [4]. As computer technology becomes more affordable and available, research into moving highly technical applications into the home is on the rise. In the field of tracking, security surveillance becomes only one of the possibilities for such a system. Others include interactive floor games, location sensitive health monitoring and information display, and "smart" control of lighting and appliances.

The algorithm described in this work presents a dynamic programming tracking methodology that relies on real-world constraints encoded using histogram intersection, distance, and time. A system implementing it was created and run in the room shown in figure 1-1, which was built inside the House_n lab at MIT. In addition to the research done in the room about people and spaces, the area is used for meetings, rest, and work by the lab members.

Contextual information is used to create a search space of possibilities that enables error recovery by considering multiple hypotheses over time. This information consists

Figure 1-1: The room viewed from the static, top-down camera

of continuity of motion and appearance, as well as knowledge about the number of people in the space and their proximity to others. The search space chooses the best decision at each step by considering all physically realizable situations over a window of time.

1.1 Background

There is an impressive body of literature on the tracking of people from visual input. Relevant works include those that use blob tracking, track multiple people, and/or retain multiple hypotheses.

Blob based tracking is used by Pfinder [23], the Closed-World Tracking(CWTA) Algorithm [10], and the EasyLiving [12] algorithm. Pfinder tracks a single, unoccluded person in a complex scene using a multi-class statistical model of color and shape to retrieve the head, hands and feet of a person. CWTA is a multi-person indoor tracker that uses a closed-world assumption to select and weigh image features, which are used for matching using a match score matrix. The EasyLiving tracker uses two sets of color stereo cameras mounted on the wall of the room to track multiple people. It uses a stereo module for locating people shaped blobs, and a person tracking module which predicts each person's next location by using a saved history of each person's path. If it finds more than one blob in the predicted space, it uses histogram intersection to pick its match. Although it retains path histories, it does not use them for error recovery.

W4 [9], by Haritaoglu et al., works outdoors with a single gray-scale or infrared camera and constructs appearance models for people after finding body parts and looking for known postures. The camera gets a side view of the space, as opposed to the overhead view used in this work. It uses second order motion models to model the movement of a person, as well as that of a person's body parts and therefore predict their next location. It also tracks each person in a group by first finding all the heads, and then tracking with correlation based matching. The algorithm, though, can be applied only to people because it depends on human body parts and motion patterns. It is also difficult to implement with an overhead camera due to self-occlusion.

Grimson et al [8] use an outdoor tracking and classification system that utilizes a set of sensors that detect moving objects, classify them, and learn patterns of activity from them.

The tracker described by Beymer and Konolige [2] uses stereo to get disparity information and then track using adaptive templates that are adapted using a Kalman filter and correlation information. A detector is used to find new people and to correct for template drift. Their approach appears to depend on a single posture, because people are found by matching to a shape model of an upright person.

All of the above trackers commit a decision on each incoming frame, and do not check whether that decision is supported in the next few frames. The algorithm presented here, on the other hand, makes decisions based on the most consistent view *over a window of time*. This approach can possibly be extended to work with other tracking systems such that when no decision is strongly supported, the system can observe what happens over a set of frames and strengthen one or the other of the available decisions. The constraints can then be changed to comply with the rest of one's tracking algorithm.

The Condensation algorithm of Blake et al. [11] keeps multiple hypothesis to track contours clutter by learning the statistics of objects dynamics and using "factored sampling" to encode possible interpretations. MacCormick and Blake provide a methodology, partitioned sampling, for tracking multiple, similar objects using probabilistic exclusion and sampling [13]. This was extended by MacCormick and Isard

to track an articulated object in [14].

Rasmussen and Hager [17] have separate trackers for different targets that use joint probabilistic data association to combine multiple attributes (color and contour) of complex objects. The exclusion principle ensures that two trackers do not pick the same target. However, they do not simultaneously track multiple articulated, non rigid, objects with similar contours.

Previous work in applying dynamic programming to tracking has mainly involved subpixelsized target tracking where DP is used to overcome the weakness of the signal [1]. It has also been used in detecting, tracking, and matching the deformable contours of a single object [7]. In both cases though, it is used to search through a set of possible locations the target may have moved to per frame.

Darrell et al. use a dynamic programming based approach to estimate trajectories over time and subsequently segment foreground regions [6]. However, adding multiple people is done iteratively thereby discarding dependencies,. This causes them to cite that overlapping trajectories cause a problem. The tracker described in this work, on the other hand, considers these dependencies and does well on people merging and splitting.

1.2 Overview

The algorithm uses data from a single overhead camera to track multiple people or objects moving around in a room. At each frame, background differencing, combined with clustering, produces foreground blobs. Then, the people known to be in the room are matched to these blobs. The system uses contextual information including the proximity of the people to each other to decide which features to update at each frame and which features to use in the matching. It extends the CWTA algorithm described in [10], which uses contextual, blob-based and closed world models to keep track of people. The salient difference is that CWTA makes instantaneous, irreversible match decisions at each frame, whereas this tracker analyzes data over a window of time while maintaining multiple hypotheses.

The algorithm creates multiple hypotheses and gathers data supporting or refuting all of them. As evidence for one hypothesis builds over time, it is used to assign people to blobs and the rest of the hypotheses are dropped. It first gathers a statistical model of the background and initializes a graph of all blob-to-person possibilities and their transition scores. At each iteration, blobs in each frame are found using background subtraction, and the blobs and last known models of people in the room are passed to the dynamic programming (DP) module which makes the best next match of people to blobs, and updates models as needed.

The DP module searches a graph of all object-to-blob match hypotheses over a window of time. In this work, this window was varied from 1 to 5 seconds. At each iteration, the best match is found for the earliest frame in the window based on all the data between the earliest frame's hypotheses and the final set of possibilities added to the graph. Each iteration shifts the window over once, adding data from the latest frame and discarding rejected hypotheses. The new data is comprised of all possible hypotheses for the frame at hand, and is linked into the graph using distance and color data to compute the likelihood of each represented hypothesis from all those derived from the previous frame. The process is then repeated.

At each match decision, the system uses the contextual information consisting of the proximity of people to each other to decide which features it should update. People's models are readable by other systems that may need to used the location information. The process is illustrated in Figure 1-2.

The algorithm uses a set of constraints in its calculation of match scores. These constraints are:

- People do not disappear through the floor of the room, and thus everyone known to be in the space will be there in the next iteration.

- People who enter a merge are the same people who leave a merge.

- Continuity in motion: people are more likely to make small motions than large ones after a small lapse in time.

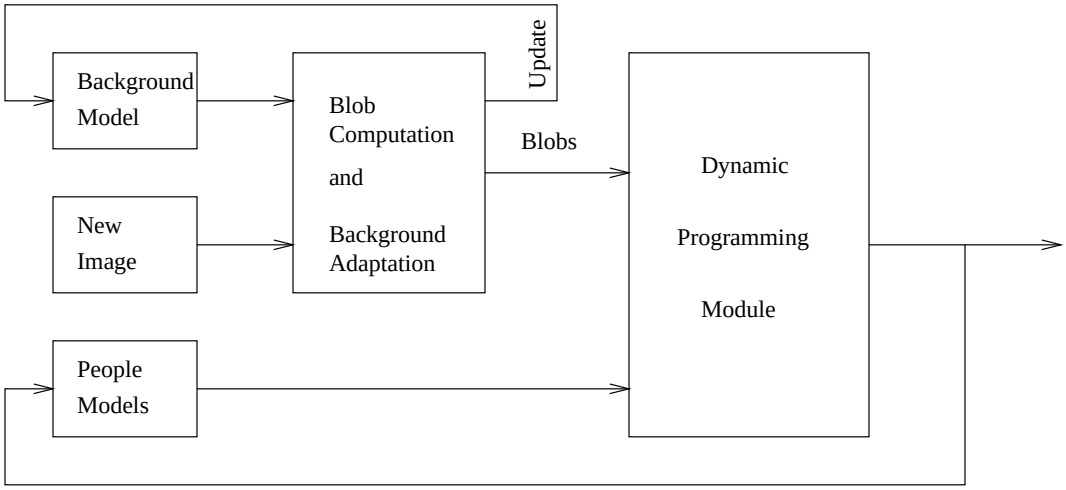

Figure 1-2: System Block Diagram

- Continuity in appearance: people look more like previous shots of themselves than shots of others.

The system is manually initialized, and tracks the people known to be in the space. A scheme for handling people exiting and entering the space, that will also allow self initialization, is discussed in Section 7.3.2.

1.3 Organization

The next chapter describes the models used for the objects being tracked, the foreground regions, and the background. Chapter 3 details the dynamic programming module, which encodes the constraints, searches the decision space, and makes the best match it can. Chapter 4 describes the histogram intersection technique used as an identity feature. Chapter 5 explains how an inter-blob, inter-frame time sensitive distance metric is calculated as a measure of motion continuity. The results of running this algorithm in a room in the lab are shown and discussed in Chapter 6. Chapter 7 discusses the effects of changing algorithm parameters, and ways to extend the algorithm.

Chapter 2

Objects, Blobs and the Background

The world and the tracked objects are modeled so that they can be located and identified in each frame. The process that uses the frames to create and update the models using the person-to-blob assignments computed by the DP module is illustrated in Figure 2-1. The tracking algorithm uses a model of the space with no one in it to subsequently locate regions that could corresponds to the tracked objects. These regions, or blobs, are found by the background differencing described below. At each time-step, each object is matched to one of these blobs.

The algorithm observes the scene for a number of frames over a window of time, as illustrated in Figure 2-1, and makes a match decision at the end of the window about the earliest frame in it. In making these decisions, contextual information is used, consisting of how many people are in the space, where they were when the last match was made, and the constraints described in the previous chapter.

This chapter describes how the background model is calculated and maintained, how blobs are found and represented, and how objects/people are modeled.

2.1 Background Subtraction and Blob Creation

Background subtraction is performed as illustrated in Figure 2-2. The background model is created when the room is empty by computing the mean and the variance of each pixel location over YUV color space for a set of frames. After the model

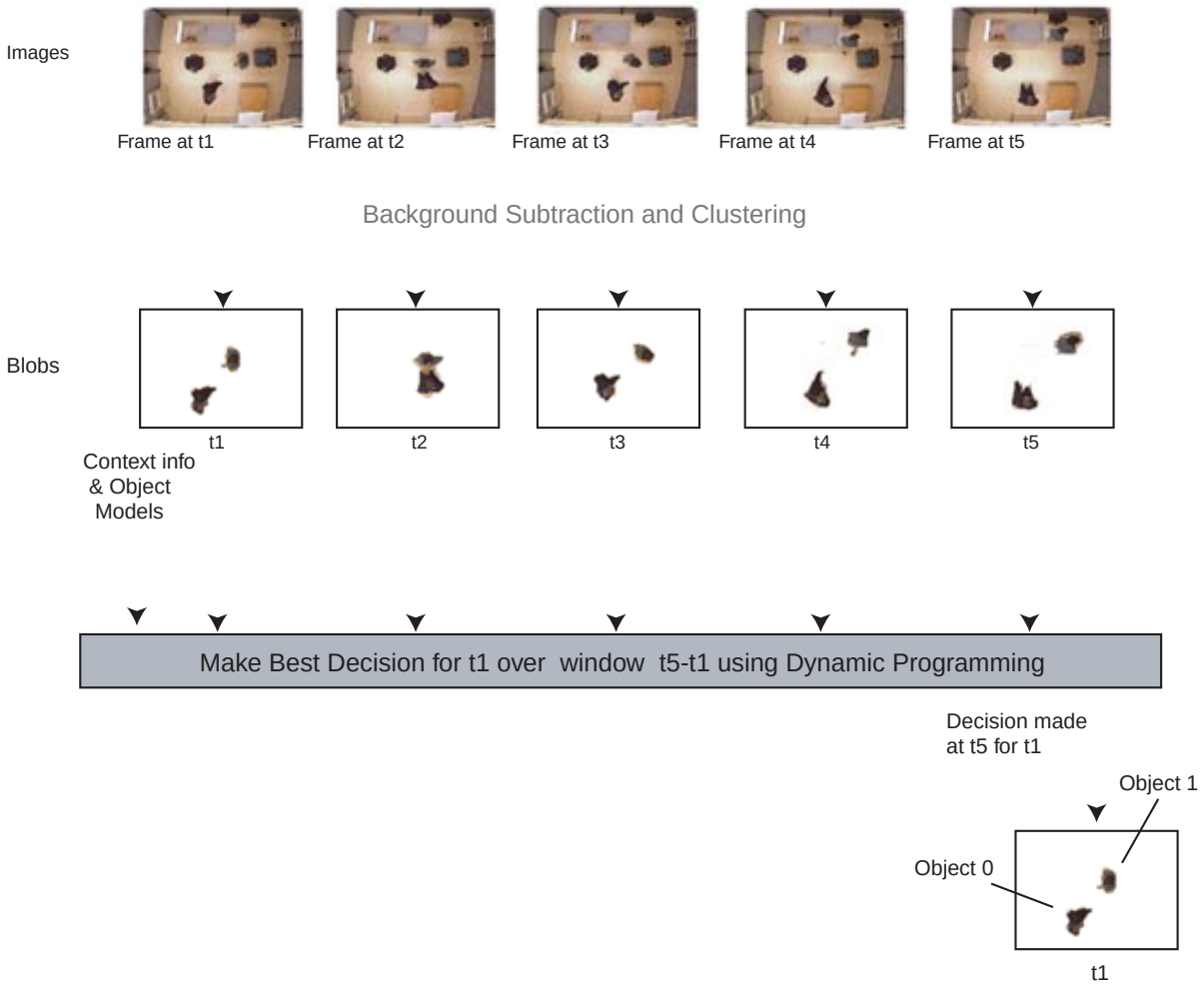

Figure 2-1: Blobs are found for each frame using background subtraction and clustering. The blobs, time, and contextual information of objects in the space are used by the algorithm to assign each object to a blob in the earliest frame in a sequence. The assignment made is such that it is the most consistent with the possibilities observed over the entire sequence and is found using dynamic programming.

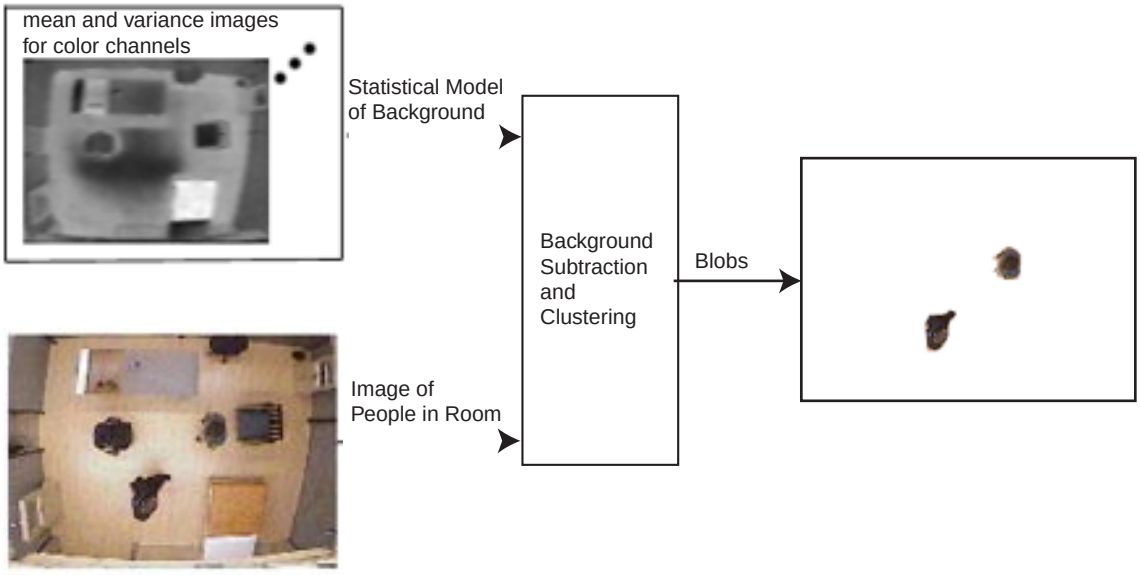

Figure 2-2: Background Subtraction takes the background model consisting of the mean and variance for each pixel over a number of frames, and a new frame from which blobs are extracted. The blobs are found by dilating and clustering pixels that deviate enough from the mean to be considered foreground.

is complete, the background is removed from all new frames using a YUV-based background differencing method. The resulting pixels are then grouped together by three successive dilation operations followed by a fast bounding box merging algorithm to combine regions that are very close to one another. The result is a set of blobs, where each blob should correspond to a single person, multiple people standing close together, or camera noise. This is done exactly as in [10].

2.1.1 Blobs

The blobs that result from the above procedure are numbered and assigned a position which is the center of their bounding box, a size based on the number of pixels, and a UV color histogram. The dynamic programming module will use this information to assign each object it knows to be in the room to one of these blobs. This assignment constitutes finding and identifying a tracked object in any frame. If two or more objects are very close to each other, the background subtraction will yield one blob for all of them.

2.1.2 Object models

Each of the objects being tracked has a position, a size, and a set of location-based color histograms as described in chapter 4 . When the algorithm assigns an object to a blob, it updates the object's model with that of the blob it found as described in chapter 3.

2.1.3 Adaptive Background

In order to run the algorithm for a long time, the background needs to adapt to changes in lighting and movement of small objects. This may be done by having a window of values for every pixel-location in the background model. At set intervals, a new frame's values are incorporated into the model and those of the oldest ones dropped from the calculation.

Thus, the background can be adapted by saving a history in each pixel-location of all the values used to calculate its mean and variance . A new frame's pixel values are added to the model by updating the mean, μ, and variance of every pixel-location in the background model such that the oldest value used is dropped out and the value from the new frame is used and saved instead.

The update can be done by updating the variance formula, where n is the total number of pixels used:

$$var = \frac{(\sum x^2) - n\mu^2}{n-1} \tag{2.1}$$

which gives the new variance that excludes x_{old} and replaces it with x_{new} to be:

$$var_{new} = var_{old} + \frac{(x_{new}^2 - x_{old}^2) + n(\mu_{old}^2 - \mu_{new}^2)}{n-1} \tag{2.2}$$

where the new mean μ_{new} is calculated from the old mean by subtracting the value of the old pixel divided by n and adding that of the new pixel divided by n.

In indoor spaces, though, people tend to stay in the same place for a while, whether watching television, taking a nap, or just sitting down to read, and the above approach would eventually incorporate the people into the background. To avoid this situation,

the background update takes the new frame, a corresponding image with the blobs masked, and which of these blobs the Dynamic Programming Module assigned to people. The background model then updates the history and values of all pixel-locations except those that have people in them. In this way, tracked objects will not be incorporated into the background.

The trade-off of using this method as opposed to simply updating each pixel is that all frames used in the window must be saved, so that the earliest one can be used for the update. The reason is that this method needs to know where the objects are, and the earliest frame in the window is the most recent one whose blobs have been assigned to objects.

2.2 Conclusion

The goal of locating each tracked object in every frame therefore becomes that of finding all blobs using background subtraction, assigning each object to a blob, and then updating the object's model appropriately. This assignment will be done by the DP module described in chapter 3 and will use appearance and distance measurements. The calculation of these measurements is presented in chapter 4 and chapter 5 respectively.

Chapter 3

Dynamic Programming Module

Inter-frame and intra-frame data are gathered at each iteration, and encoded to enforce the real-world constraints the system is based on, as described in section 1.2. The algorithm tries to assign people known to be in the room with the blobs in each frame, in keeping with the knowledge that they cannot simply disappear through the floor.

Continuity in appearance is encoded using the color histogram information between a person's model and the frame in which the person is being located. Here, the system uses the idea that a person will look more like he or she used to when it could get a good shot of them than like the other people.

Motion continuity is checked using the distance between blobs in consecutive frames. The blob in one frame that is closest to a blob from the previous frame most probably matches to the same person because people cannot go from one end of the room to the other in a small amount of time.

Using distance in conjunction with color data, the algorithm is able to continue tracking people during and after merges. When a merge occurs, multiple people appear as one blob, and there is no proper segmentation of individuals to give clean histograms of each. Instead, the result is a histogram of all of them together. The distance score helps here because of the constraint that people who enter a merge are the same people that leave a merge. Even though the system can't pick up a good color match, it knows who is in the merge because measures were good as the people

came closer, and the distance supports them being in a merge even though the color match is weak. As the people separate, the distance score is nearly the same for all of them, but the histogram is able to discriminate between them because now they are isolated again. Thus, the two measures complement each other such that when one measure cannot provide strong evidence as to who is where, the other one can.

The dynamic programming module puts this data together and searches it such that the most coherent decision can be made at each iteration.

3.1 System Structure

At each time-step, the algorithm matches each object known to be in the room to one of the blobs that the background subtraction yields for a specific frame. Match decisions are made over a window of time. The window includes all possible people-to-blob matches, saved in an edge-weighted graph in which the edge-weights are transition costs between window frames. These edge-weights are calculated to enforce the constraints described above. The structure of the process used is shown in Figure 3-1 and is described in detail below. The figure shows an example of window size 5 and two people and two blobs in each iteration.

The blobs are found in each step and their histograms, locations, and sizes are saved. All possible matches, with repetition, are generated of the blobs to the people in the space. These possibilities are added as different states in the last stage of the graph, as is shown by the "add new stage" arrow in Figure 3-1. Transition costs are calculated for going from any possible state at one time step to all the possibilities in the following time step. The resulting graph is then searched for the shortest path from the start node to any node in the final stage using dynamic programming over a window of time whose size is the number of time steps saved per search. Due to the structure of the graph, the search will yield the best path, illustrated by the gray nodes in the Figure 3-1 after the "DP for Shortest Path" transition, given the constraints described in section 3.

The start node represents the last decision committed, and the search yields the

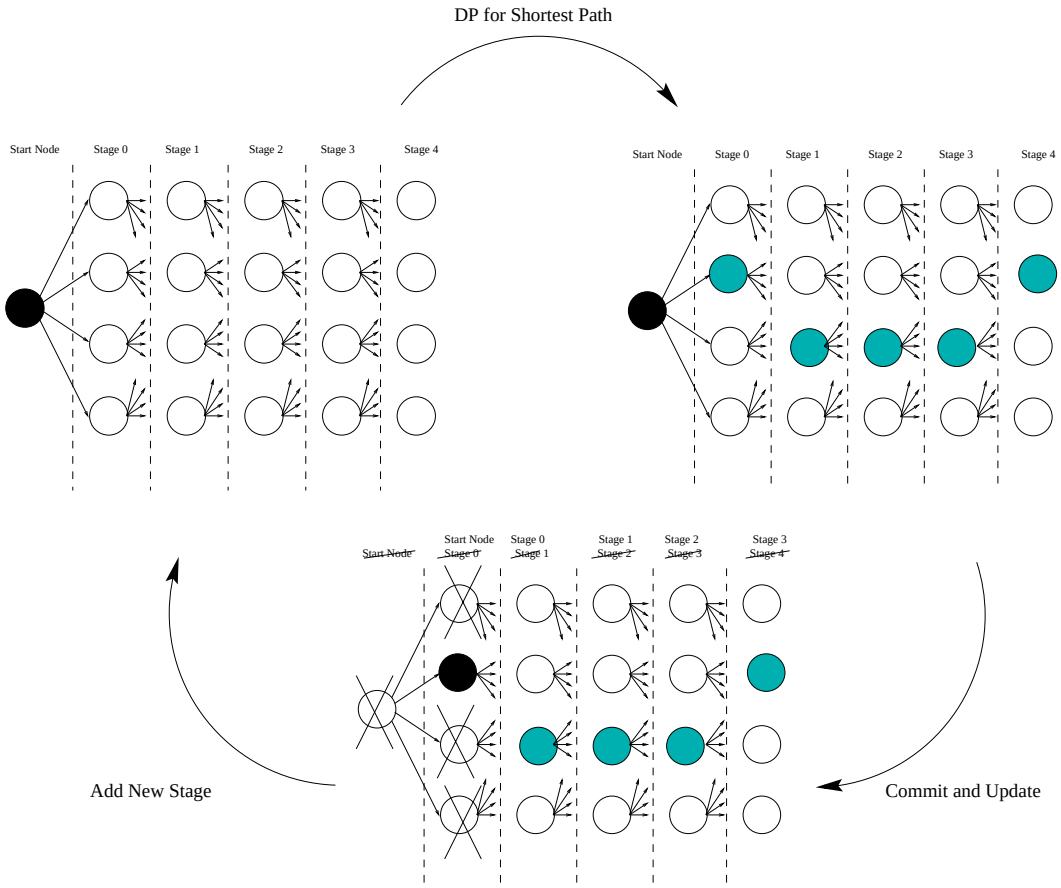

Figure 3-1: DP Module "search-commit-add stage" Cycle

best next state to go to from there, where each stage indicates a possible set of people-to-blob assignments. The chosen next node's person to blob assignment is then committed and it becomes the new start node, as shown by the "commit and update stage" in Figure 3-1. A new stage is then added, and the "search-commit-add stage" cycle is repeated.

3.1.1 The Nodes

Each node represents one possible complete state of the world. Figure 3-2 shows the structure of a single node in the graph at stage t. The state of the world is saved in the node's state, and consists of an assignment of each person in the room to one of the blobs in the room at the stage considered. Multiple people may be assigned to the same blob, which occurs if the people are standing close enough together. Each node

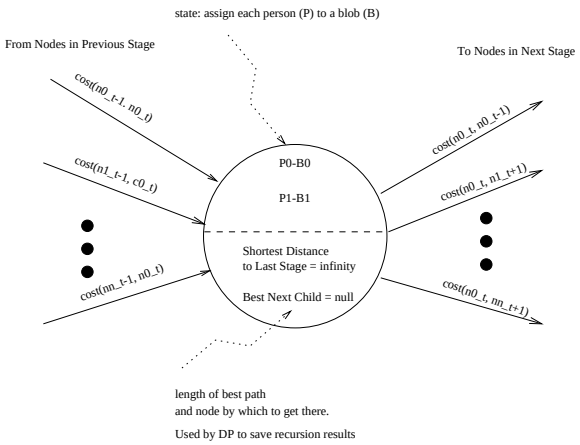

Figure 3-2: Close-up of a Node: State shows the person to blob assignment. Transition costs are associated with each link coming into and leaving the node. All links come from the previous stage and go to the next stage. The recursion results of dynamic programming are saved in each node as the length of the shortest path to the final stage and the next node in that path.

is linked to those from the previous and next stages, with an associated transition cost for each link.

3.1.2 The Graph

The graph represents all possible person to blob assignments over all the stages, with transitions from all the nodes in each stage to all those in the one after. The nodes in the newest stage have no children. The number of stages in the graph corresponds to the size of the window of data the algorithm will look at before making any decisions. The number of nodes per stage corresponds to the number of possible ways to match the people in the room with blobs in the corresponding frame. This number, as well as the number of links, can be reduced by performing distance thresholding as described in section 5.3.

A stage is simply the representation of an analyzed frame: each time a frame is analyzed and its data inserted into a graph, that data forms a set of nodes that are in the same stage. One may also think of it as a time-step, but since frames do not necessarily have to be analyzed at constant time intervals, the term "time-step" may be misleading.

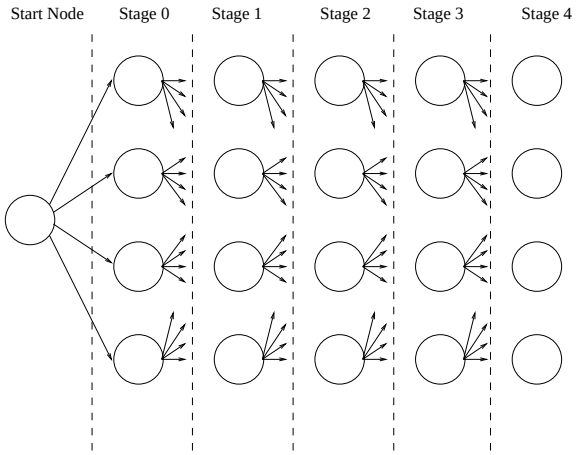

Figure 3-3: Graph structure: Example with window size 5, with two people and two blobs in each iteration.

A sample graph of window size 5, with two people and two blobs in each iteration, is illustrated in Figure 3-3. The start node represents the last committed decision. From that point, any transition into stage 0 may be made. An important characteristic of this graph is that all nodes in the same stage may only have nodes in the subsequent stage as children and nodes in the previous stage as parents.

3.1.3 Calculating Transition Costs

At each iteration, transition costs are calculated to all the nodes at the new stage from those in the previous stage. The transition cost between a parent node p and a child node c is calculated using the person to blob assignments found in the state of the nodes. The cost encodes the constraints described in Section 3, which are estimated in this implementation using distance, histogram intersection, and time.

Decisions are delayed by the size of the window used. Blobs distances are based on how far they moved from stage to stage, and consist of distances from all the blob at stage $t-1$ to all the blob at stage t. This distance is used instead of that from the person's model to the new blobs because a number of stages come between the person model's update and the stage containing the newest blobs. The histograms, on the other hand, are calculated within a single frame, by comparing each new blob's histogram to those in the person models.

For each node the cost is equal to the sum of the sums of normalized distance and histogram scores, as well as a weighted merge cost. The distance score for $person_i$ is a time sensitive measure based on the normalized Euclidean distance from the blob matched to $person_i$ in the parent node (previous stage) to the blob matched to $person_i$ in the child node (newest stage). A complete description of its calculation is in chapter 5. The distance score incorporates time so as to encode the constraint that people do not instantaneously move across the room. The function used to calculate it in this implementation is linear if the Euclidean distance is within an allowable range for the given inter-stage timing and a sigmoid function if it is above that distance. The allowable distance is dependent on time, and thus large distances are heavily penalized if the inter-stage time is short, and lightly penalized if it is long.

The histogram score is the result of the intersection of the histogram of the blob matched to $person_i$ in the current node, and the histogram saved in $person_i$. This calculation is detailed in chapter 4.

The merge cost is a constant used to penalize the system for assigning multiple people to the same blob. Otherwise, the system will prefer to make such assignments when they are incorrect. For example, consider the case when two people meet, and then one of them moves quickly away but the other remains in the same position. The distance score is then practically zero if both people are said to have not moved. Therefore, the algorithm will want to choose that even if it gets a lower histogram match. The merge cost will tip the balance to let it match to the blob that has moved away.

The merge cost is weighted by the difference between the number of people said to be assigned to the same blob and the total number of merges in the state. For example, the state $P_0-B_0, P_1-B_1, P_2-B_1, P_3-B_3$ will have $1M$ added to all transitions to it, while the states $P_0-B_0, P_1-B_1, P_2-B_1, P_3-B_0$ and $P_0-B_0, P_1-B_0, P_2-B_0, P_3-B_1$ will both have $2M$. The cost is then normalized by the number of people in the room.

Therefore:

$$cost(p,c) = aM + \sum_{i=0}^{numPeople} (f(D_{B_{i,p}-B_{i,c}}, \Delta t) + H_{(hist(B_{i,c}),besthist(P_i,B_{i,c}))} \quad (3.1)$$

$$cost(p,c)_{norm} = cost(p,c)/N \quad (3.2)$$

where:

- $cost(p,c)$ = cost from node p at stage $t-1$ to node c at stage t
- $P_i = i$th person
- $B_{i,c}$ = blob matched to $person_i$ in the child node
- $B_{i,p}$ = blob matched to $person_i$ in the parent node
- $besthist$ returns the closest histogram of P_i to the location of $B_{i,c}$
- H calculates the histogram intersection which goes from 0 to 1 where 0 is best
- D gives the distance between two blobs and is calculated as described in section 5.1
- a is the total number of people merged minus the number of merges
- M is a constant merge cost whose choice is described in section 6.4.1
- $f(d,t)$ is the time-based distance function between a blob in stage $t-1$ and a blob in stage t as described in section 5.2 and plotted in figure 5-2.

3.2 DP Search For Best Path

After initialization as described above, a dynamic programming algorithm is run on the graph to find the least cost path. The problem is one of finding the best policy for a setup that consists of a fixed number of stages, each having a finite number of nodes. Each node can store its best distance to the goal, as well as through which child that distance was achieved, as shown in the node illustration in Figure 3-2. All

the nodes in the final stage have their best distance to the goal be zero, all others start with it set to a very large number. The algorithm will find the best path from the start node to any of the nodes of the final stage. The problem is similar in structure to the "Stagecoach Problem" in [18].

At each node, the best path is the minimum, over its children, of the sums of the transition cost to a child and the child's best distance to the goal. An important property of the graph used is that all the children of a node are in the stage just after the one the node itself is in. The recurrence relationship which defines the best path to choose from a node i is therefore represented by:

$$c_n^*(i_n) = min_{i_{n+1}}\{c(i_n, i_{n+1}) + c_{n+1}^*(i_{n+1})\} \tag{3.3}$$

where:

- n goes from the $S-2$ to zero, S is the number of stages in the graph, and $S-1$ is the last stage in the graph
- $c_n^*(i_{S-1}) = 0$ all the nodes at the final stage have zero cost to signify reaching the end
- i_n is the ith node in stage n
- $c_n^*(i_n)$ is the cost of the optimal path for the i^{th} node in a stage when there are $number of stages - n$ stages remaining
- $c(a, b)$ is the transition cost from node a to node b
- min_{i_n+1} gives the minimum over all the nodes in the next stage, which are all the children of i

At each iteration, the child that gave the best path is saved in the node along with the best cost, $c_n^*(i_n)$, from that node until the end. The algorithm runs from the final stage of the graph to the start node, and every time it sees an unvisited node it clears the values of the best child and cost in case they are from a previous call and overwrites them with the new value found. Therefore, every time an optimal

path from a node needs to be calculated, one can find all the optimal paths from that node's children by simple lookup. Once the algorithm terminates, the cost of the best path is saved in the start node, and the path itself can be found by traversing the graph from the start node, and going down the saved best child of each node reached.

Only the best child of the start node is used to commit the decision. At the time that it is committed, it is the node whose choice is supported by all the data gathered so far. All subsequent nodes in the path are supported by less data and may change once a new stage is taken and the dynamic programming algorithm is run again.

3.2.1 Example

Consider the case when two people merge, and then one of them walks away while the other stays in the same location as illustrated in the top level of Figure 3-4. For the first few stages taken after the split, the lowest cost path will be obtained by keeping both people merged with the person that did not move due to smaller distance scores. Eventually, though, a new stage might make cause the path deciding to keep them merged to become too expensive due to the accumulation of the merge cost. The best path will then shift to going through the unmerged blobs as it should. If this occurs before the first mistaken decision reaches the start, it will never commit the mistake, as is shown in the lower layer of Figure 3-4.

3.3 Committing Decision and Updating Graph

Once the dynamic programming algorithm picks out the best node to go to from the start node, the system commits the person-to-blob assignments based on that node's state and subsequently updates the graph. The commit involves copying data from a blob to the person assigned to it. The data copied depends on how close two people are said to be from each other, and how long it has been since a person was last merged. The pseudo-code below concisely describes how a person's model is updated once it is to be assigned to a blob on a commit step. The $need\_Hist$ function is shown in chapter 4.

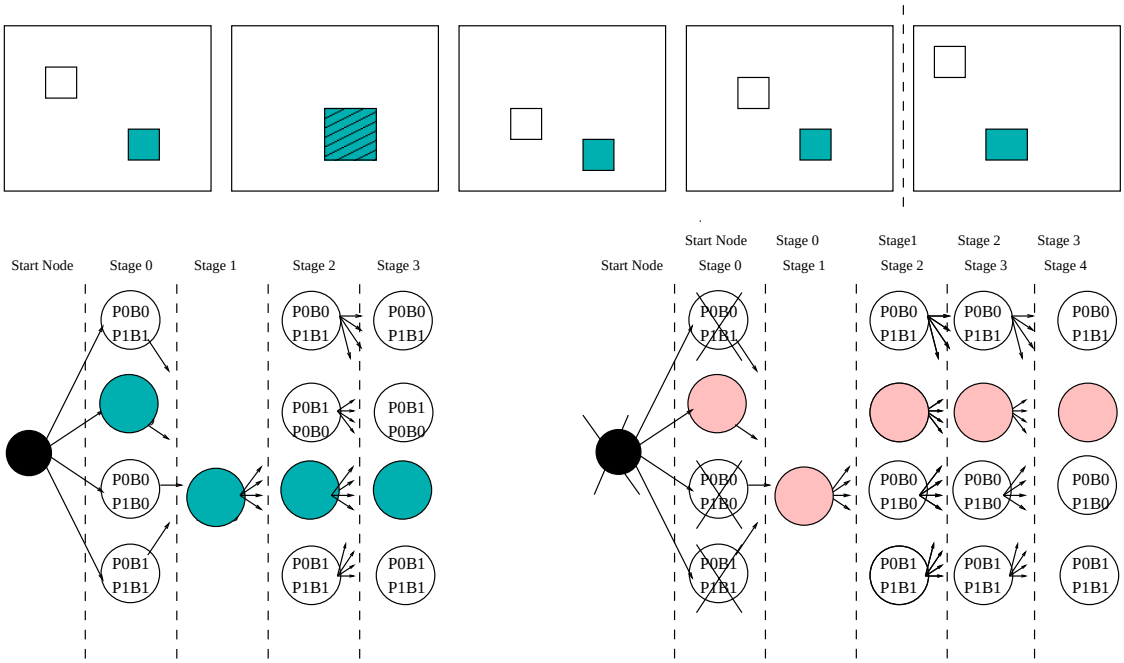

Figure 3-4: TOP: two people in the room correspond to light and dark boxes. The people merge and then separate, with one moving away while the other stays in the location of the merge. BOTTOM: Example corresponding graphs,each column corresponds to a frame. The first and third row represent a merge, and the second and third rows represent one person to one blob assignments. For the first four frames, DP may find the least cost path to be one that people merged for 3 steps instead of 1 (left graph). In the next iteration, with the 5th frame, DP adjusts the path to only one merge step(right graph).

```
model update:

1.loop through commited node state
2.  set person as assigned to blob
3.  count number of people assigned to same blob
4.  mark those people as merged

5.loop through all people
6.    person.position <- oldestBlobs[person.blob].position

7.  if person is in a merge
8.  then  timeOfMerge <- timeOfOldestFrame
9.  else
10.     loop through all blobs
11.       if the person's blob's bounding box intersects
              any other blob's bounding box
12        then farEnough = false

13.      if (timeOfOldestFrame - timeOfMerge > 2000ms
             && farEnough)
14.          then  if(person.needHist())
15.          then
             person.setHistogram(oldestBlob[person.blob].histogram)
```

If two or more people are assigned to the same blob (merged), the model update involves copying only the position of the blob. On the other hand, if a person is far enough from all other people in the room, then the size and possibly the histogram of the blob are copied into its model as well as the position. Whether or not the histogram is copied in this case depends on whether the model has a histogram closer to the center of the area the person is in, as described in Chapter 4. Each person

Figure 3-5: Segmentation when people are close to each other: The left picture shows a merge where people are lumped into one blob. The right picture shows unreliable segmentation due to proximity. The squares shown are the bounding boxes of the blobs found in the image.

has nine histograms corresponding to different areas in the room. When a histogram is about to be copied into a person's model, the blob's location is checked and the person's histogram corresponding to that location of the room is overwritten.

Two cases were considered to take into account steps when people are really close to each other. In both of these cases, histograms are not updated, and are checked for in lines 11 and 13 of the pseudo-code above. The first case is for a set time after the person has been in a merge (currently set to 2 seconds). The second case occurs when a person is assigned to his own blob, but that blob is so close to another one that their bounding boxes intersect.

The reason for these cases is the instability of the data when people are so close to each other. The segmentation is not always perfect and may put one person's hand in another person's blob for example. During the time that people are next to each other, they tend to move back and forth a little so that the system either lumps them in one blob, or cuts up different parts of them into separate yet not very reliable blobs due to proximity and shadows, as can be seen in Figure 3-5. At this time, the position data is quite accurate, but the histograms are not, and neither is the assignment of multiple blobs that are very close to each other. Waiting until the blobs are reasonably separated allows for a clean enough shot of each so that reliable histograms are computed.

The main constraint here is that people who enter a merge must be in the merge until they are seen leaving it. Therefore, if a person is determined to be in the merge, he must still be within the noisy blobs near the merge location.

The "commit and update" stage in Figure 3-1 provides an illustration of the changes that occur in the graph. After making the update to the models of the people in the room, the system needs to make space for a new stage. The node just committed becomes the new start node, and all its siblings as well as the old start node are removed from the graph. The graph is now of size one less than the window, and the next generated stage can be added at the end so the next decision to be made can be found.

3.4 New Stage Generation

Every time a frame is analyzed, a new stage is added to the end of the graph. This process is what occurs in Figure 3-1 in the "add new stage" transition. All the blobs produced by background subtraction on that frame are used to generate the new nodes. Thus, the number of nodes generated per stage is b^p, where b is the number of blobs in the image and p is the number of people known to be in the room. For each node, every person gets assigned a single blob, with extra blobs ignored if necessary. The assignments are then saved in the node's state, and it is added to the stage being created. Then, all nodes from the previous stage set it as their child with associated transition costs. When all nodes have been generated and linked in this way, the stage generation process is complete. Since the combination generation is exhaustive, the nodes at any stage describe all possible scenarios for the analyzed frame.

Every time a new stage is saved, all blob data from that stage is saved in a matrix the same width as the graph with each column corresponding to a different stage's blobs. The data saved includes position, histogram, and size. The mask image for the new stage is saved as well, so that it can be used in the distance calculation as described in chapter 5.

All distances from blobs in the previous stage to those in the new one are calcu-

lated, normalized, and saved. Histogram intersection scores between every person in the room and every blob are calculated and saved as well. These values can then be simply looked up when transition scores are being obtained as described in section 3.1.3.

3.5 Initialization

In the current implementation, no decision is made without evidence, causing the number of stages in the graph to be equal to the window size before a decision is committed. The system is started with a graph consisting simply of a start node with an invalid state. Each time a frame is analyzed, a stage is created as described in Section 3.4.

The start node points at all the nodes of the new stage, but since no histogram and no distance differences can be calculated at this point, the transition costs are set to prefer a particular child by being very high for all its siblings. When these links are created, the preferred child's state is used to assign the first histograms to the people. The first transition is thus forced to the node where all people are assigned to separate blobs. The initializing constraint used to create a proper initial assignment in this manner is that the system starts up with at least one frame where everyone is in the room and sufficiently far from everyone else.

Alternatively, if all new nodes of $stage_0$ are given an equal cost from the start node, the system would prefer to assign all the people to a single blob and follow that around until the merge score accumulates and forces another option. Therefore, performance is much better when the initialization step is forced as above.

A scheme for handling entering and exiting is discussed in section 7.3.2 and may be used instead. This scheme would create a new person model when a new person walks in and delete the appropriate model when a person exits. Thus, there would be no need to force an initialization.

Until the required graph size is reached, no decisions are made and all new stages get linked as usual, with transition costs. If a person has no histogram, which could

occur if a scheme is used for people entering the room, a non-partial histogram score of 0.5 can be used.

When the required dynamic programming window size is reached, the system is ready to run the dynamic programming algorithm that will find the least cost next node in light of the data in the entire graph.

In order not to compute all possibilities at each iteration for a new stage, the system initializes a table that has all possibilities for any number of people (up to a given number which is the maximum) and any number of blobs(up to a given maximum as well). Then, when each stage is generated, the algorithm performs a look up in the table based on the number of people and the number of blobs, gets all the possibilities, and maps them to the current valid people and blobs. The mapping is done because in the implementation, a node's state contains place holders of up to the max number of people, such that all free models (more than the number of people actually in the space) are matched to the "invalid blob" -1 and are therefore ignored.

3.6 Complexity

At each stage there are b blobs and p people. Therefore, at each stage, there are b ways of assigning each person, yielding a total number of nodes per stage equal to b^p.

If N is the number of nodes in the graph except the start node, k is the number of stages, and E is the total number of edges, each stage has N/k nodes. Therefore, the number of edges per stage is $(N/k)^2$. The last stage has no children, while the start node has N/k children.

This implies that the total number of edges is

$$E = (\frac{N}{k})^2(k-1) + (\frac{N}{k}) \tag{3.4}$$

As N gets large, and is much larger than k, the number of edges is $\mathrm{O}(\frac{N^2}{k})$ where $N = k(b^p)$

Therefore, the number of edges is $\mathrm{O}(kb^{2p})$

The dynamic programming algorithm traverses each edge once and does one addition per edge. Therefore the running time is $O(E) = O(kb^{2p})$).

In section 5.3, a scheme for distance based pruning of the graph's nodes and links will be presented that will substantially decrease the complexity for stages when people are far apart and the inter-stage distance is less than one second. Chapter 7 presents a possibility for splitting the tracker into two graphs and combining the results in case more people than can be handled enter the space. The drawback of this approach is that dependencies between objects will be lost.

Chapter 4

Histograms

4.1 Introduction

In looking for a method for measuring identity matches based on appearance, the main issues are size variation, multiple views, imperfect segmentation, and execution speed. As people move about the space, the size of the blobs picked up will vary depending on what they are doing and where they are. Each time a frame is taken, the same person in a different location is seen from a new viewing angle. Based on these factors, the technique of Histogram Intersection is very well suited to this application.

Histogram Intersection gives a match score of how similar objects are based on their color distributions. Swain and Ballard [19] show that histogram intersection is robust to multiple views, scales, occlusions, and distractions in the background. It is also linear in the number of elements in the histograms. One drawback of the algorithm is that it is not robust to changes in illumination.

Two changes were made to the Swain and Ballard algorithm. One change compensates for the illumination dependency by using UV color representation instead of the RGB representation used by Swain and Ballard. In order to further strengthen the model, multiple histograms are assigned to each person based on where they are in the room.

4.2 Histogram Intersection

A histogram of a section of an image is obtained by counting the number of times each color appears in the image, with the color axes divided into large buckets. One can calculate the similarity between two objects by computing a match score for their histograms. Given a model histogram M and an image histogram I with n buckets each, the algorithm returns the number of corresponding pixels of the same color that are found between the image and the model. For a score between 0 and 1, the score is normalized by the number of pixels in the model. The histogram intersection of two histograms is therefore defined in [19] as:

$$\mathcal{H}(I, M) = \frac{\sum_{j=1}^{n} min(I_j, M_j)}{\sum_{j=1}^{n} M_j} \tag{4.1}$$

4.3 Modifying Histogram Intersection

The histogram of an image region in calculated using the pixels of the blob inside a blob's bounding box. The image histogram is scaled to the size of the model histogram as recommended by Swain and Ballard in order to get a scale invariant score. The result is then subtracted from 1 because the system tries to minimize cost. Therefore, the equation for the histogram intersection score used in this algorithm is:

$$H(I, M) = 1 - \frac{\sum_{j=1}^{n} min(I_j \frac{S_M}{S_I}, M_j)}{\sum_{j=1}^{n} M_j} \tag{4.2}$$

where S_M and I_M are the sizes of the model and image histograms respectively and are defined as $S_M = \sum_{j=1}^{n} M_j$ and $S_I = \sum_{j=1}^{n} I_j$.

4.3.1 UV Color Representation

UV color space with sixteen U bins and sixteen V bins, instead of the RGB space described in [20] is used to compensate for the illumination sensitivity of histogram intersection. Although the system runs with the same lighting, illumination varies in the room depending on distance from the light source and the reflectivity of the

Figure 4-1: The Nine Areas Of The Room

room's surfaces. Therefore, the people change in brightness as they approach the areas of different illumination.

4.3.2 Location Based Histogram

Each person's model contains nine histograms corresponding to the areas obtained by dividing the room into nine equal areas, as illustrated in figure 4-1. These histograms serve two purposes: first, they provide area specific models of the people, which is useful because the segmentation may pick up different parts of the person when they are in differently lit areas of the space. Second, they allow the algorithm to recover from an error if two people are swapped and then go next to each other in an area were the histograms have not yet been swapped. A similar idea of splitting the room into a grid is used in the EasyLiving tracker [12] to make up for illumination dependencies. To get the best spread of values, the system tries to get histograms close to the center of each area. If the tracker decides that a person has matched to a blob and that it has a clear shot of that blob, it will try to update the person's histogram data with the blob's. The system checks which area of the grid the new histogram is in:

```
getHistogramArea(position)

areaX <- floor(position.y/areaHeight)
areaY <- floor(position.x/areaWidth)
area <- areaY*areaWidth + areaX
```

If it is does not have a histogram for that area, or if the new histogram is closer to the area's center than the one already in the model, it saves the new histogram in the corresponding area's slot. Otherwise, the new histogram is rejected. This variable update is illustrated in the pseudo-code:

```
Histogram update:
1.if(person.needHist())
2.    then  area <- index of grid area found from position
3.          person.histogramArray[area] <- copy of new histogram
```

```
needHist():
1.area <- getHistogramArea(position)
2.areaCenter <- center of that area found from width, size, and index of area
3.if(histogramArray[area] is empty)
4.  then return true
5.else
6.  oldPoint <- point where last histogram saved for this area was taken
7.  if(distance(currentPosition, areaCenter)<distance(oldPoint, areaCenter))
8.     then return true
9.   else return false
```

When a blob is being matched against a person's histogram model, the system will try to use the model histogram that corresponds to the blob's location, or the nearest one to it as described in the pseudo-code below.

```
getBestHistogram(position):
1.if !person.hasSavedHistograms()
```

```
2.   then return null
3. else
4.    area <- index of grid area found from position
5.    if person.histogramArray[area] is valid
6.      then return person.histogramArray[area]
7.    else
8.        loop through all areas, A[i], that have a histogram:
9.          find distance from position to center of A[i]
10.         save area corresponding to smallest such distance
11.       return histogram saved in that area
```

If no histogram has yet been saved for that location, the system uses the one in the nearest of the nine areas to which the person does have a histogram(lines 8-11). If no histograms have been saved for that person, this process returns null (lines 1-2) and the score calculation will give an impartial score of 0.5.

The next chapter explains the calculation of the distance metric, which is factored into the graph with the histogram score as described by equation 3.1.

Chapter 5

Calculating Distance

Inter-blob distances are used by the tracker to maintain continuity of motion. If a blob A in a frame is a very small distance away from a blob B in the next frame, and far from all others, then A and B most probably match to the same person. This algorithm, though, may take frames at different times, and as that time increases the distance a person may travel will increase as well. Therefore, the distance metric used is time-dependent.

Obtaining the distance metric is a two step process: first, the actual inter-blob distances for the two frames are measured, and second, a time-dependent metric is calculated from that distance.

The distance was measured by a computationally efficient approximation of the shortest distance between the contours of the two blobs, with a bias in case of overlap, as described in section 5.1. This was chosen over the distance between the centroids because the latter may be too large to be considered possible after merges as is described in section 5.1.

The time dependency was achieved by using a linear to sigmoid function described in Section 5.2. The distance metric is linear with the measured distance if it is within the allowable distance traveled for the inter-frame time given. After that, the metric rises quickly with the sigmoid, effectively disallowing matches that require objects to move too far in a given time.

5.1 Measuring Distance

To calculate the transition cost from one stage to another, the algorithm measures distances from each blob in the frame of the first stage to each blob in the frame of the second.

The distance used is an approximation of the shortest distance between the contours of the blobs. With the high penalty on large moves discussed in section 5.2 and the segmentation used that puts people who are close in one blob, this is better than the inter-centroid distance of the two blobs. The inter-centroid distance considers blobs as point masses and does not discriminate between large, merged blobs, and small ones. When people are merged people, represented by a very large blob, and a person leaves it, that person's blob will be very far away from the merged blob's centroid. This distance will be heavily penalized. Therefore, the inter-centroid distance makes moves out of a merge too costly to be considered even with the use of the merge cost described in chapter 3. However, the person is close to the edge of that blob from which he exited, making the inter-contour distance a more logical choice.

This approximation was chosen for its efficiency advantage over relatively expensive algorithms that compute actual closest points between two blob boundaries. It requires saving the two mask images which corresponding to the two stages. The mask images have all pixels in the blobs masked out, and are used to find the edge of each blob. Three possibilities are then considered, and are illustrated in figure 5-1:

1. The blobs do not overlap, in which case the distance measured is the sum of a non-overlap penalty (currently 5 pixels, or 16.6cm) and an approximation of the shortest distance between the contours of the two blobs.

2. The blobs partially overlap, in which case the distance measured is a partial-overlap penalty (currently 2 pixels or 6.6cm)

3. The blobs completely overlap, in which case the distance measured is zero. This is found by checking if the bounding box of either blob is enclosed in that of the other.

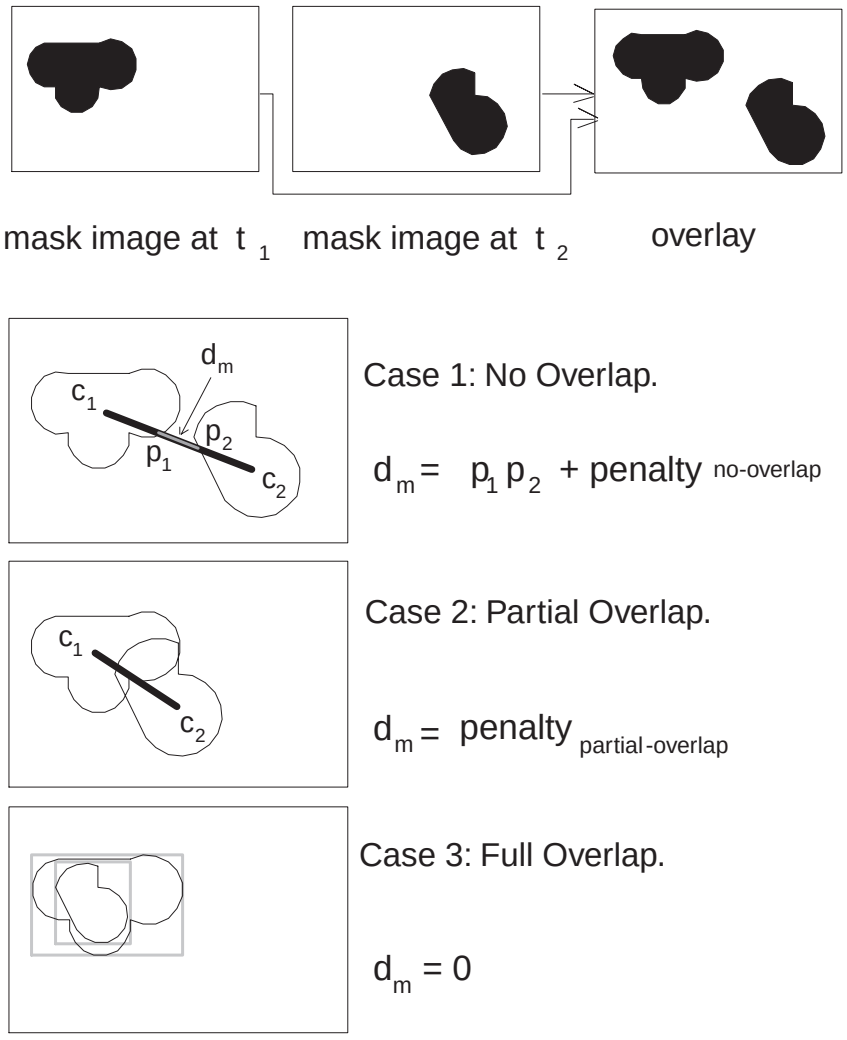

Figure 5-1: Distance Calculation Cases: TOP ROW:the mask images from two frames containing one blob each. The distance between them is calculated by considering them to be in one overlayed frame as shown on the right. BOTTOM: three different cases of distance calculation, based on the relation of the blobs to one another. Only the contours of the blobs are shown for clarity.

The first two cases are found in the same manner. The idea is to find the distance between the two pixels on the blob contours that are on the line joining the centroids. First, assume the two blobs are in the same frame, and find the equation of the line joining the centroids of the two blobs. The segment joining them is then traversed, pixel by pixel starting from the centroid c1 of the first blob. Both mask images are checked at each step along the segment for a non-masked pixel. If a pixel, P_1, that is not masked in both images is found, the process is repeated from the centroid of the other blob to find the equivalent P_2. The distance added to the non-overlap penalty in the first case (no overlap), is the Euclidean distance between P_1 and P_2. This is equivalent to traversing the line on an image where the two mask images are overlayed and checking for the unmasked pixel in that single image.

On the other hand, if the segment is traversed and no unmasked pixel is found, then the two blobs overlap (case two above) and the distance is zero plus the partial-overlap penalty.

The resulting distance is then handed to the time-dependent function described next and shown in figure 5-2.

5.2 The Time-Dependent Distance Metric

The time-dependency centers around an upper bound of the average velocity of a person in the space. Blob moves below this velocity receive a metric that is linear with the actual measured distance between them. However, faster moves are penalized severely, as discussed below.

The velocity used here was approximately 1.45 meters/second. This velocity was chosen as an under-estimate of 1.6m/s, which is cited by medical literature [15] as the "most efficient self-selected walking velocity for non-amputees".

In our space(4x6m, 160x120 pixels), this was equivalent to moving across the room in five seconds. The under-estimate on the velocity is due to the chosen distance measure, which returns a value less than the actual distance traveled because it measures from the contours.

The time between two frames is used to get the allowable distance traveled D_a from the allowable velocity V_a as per the velocity equation: $D_a = V_a/t$.

If the distance measured, d_m, by the procedure described in section 5.1 is less than or equal to D_a, then that is what is returned. Otherwise, the measured distance is scaled by the allowable distance and substituted into the penalty sigmoid function below and illustrated in figure 5-2.

$$f(d_m, t) \begin{cases} \frac{d_m}{t*V_a} & \frac{d_m}{t*V_a} <= 1 \\ \frac{50}{e^{\frac{(\frac{-d_m}{t*V_a}+2)}{0.15}}+1} + 1 & otherwise \end{cases} \tag{5.1}$$

The resulting metric will be slightly higher than 1 for distances slightly above what is allowable, and will then rise quickly until it plateaus at 50. The choice of the sigmoid was governed by finding a function that will allow a few values that are around the allowable distance, and then get too high to be considered. The plateau was chosen so the sigmoid would have a smooth curve around the value 1, but still penalize heavily for double the allowable speed (value 2). The sigmoid was translated by 1 up the y-axis to give a smooth transition from the linear 0-1 penalty function to the sigmoid for values greater than 1. The result will disallow large moves in short times, therefore eliminating swapping people who are on opposite sides of the room.

On the other hand, if the time between stages is large, the allowable distance will increase, making large moves possible. This then makes the histogram score take the bulk of the decision making because people could have moved anywhere since the last frame was analyzed.

5.3 Distance Pruning: Effect on Complexity

If inter-frame times are reasonably small (less than one second), then people walking far away from each other will have blobs that cannot possibly be swapped. Therefore, the transition cost to nodes that merge or swap them may be about an order of magnitude higher than to those that don't.

When linking the new stage to the graph, a high threshold can be put on the

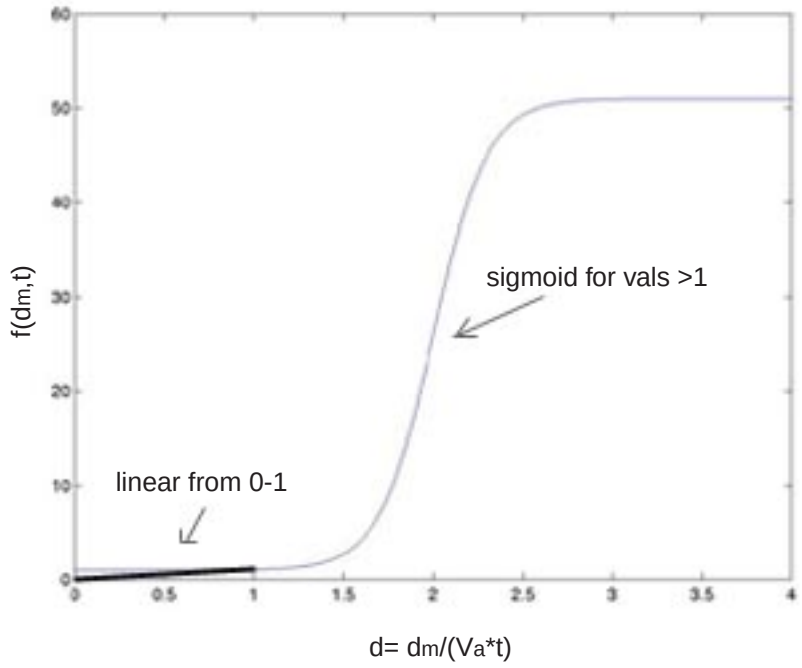

Figure 5-2: f(d, t): Time based distance function. The x-axis is $d_m/t * V_a$ such that 1 corresponds to the allowable distance moved for a given inter-stage time. The y-axis shows the distance metric returned which is used to calculate part of the transition cost function given in equation 3.1.

Sequences	Min. edges/stage	Max. edges/stage	Ave. edges/stage-Sequence
2 people	1	36	6.8
3 people	1	1951	235.9
3 people-noisy	1	3447	543.9
4 people	1	50047	4290

Table 5.1: Reduction from Distance Based Link Pruning for distance metrics greater than 8. Pruning only the links based on distance gives a substantial reduction in the number of edges per stage, as can be seen above. Numbers are shown on the minimum, maximum and average number of links per stage when pruning was done. Although the maximum is quite high, the average is sustantially less than the worst case. The sequences marked noisy had about 2 blobs more than peple in most runs. The 4 people run shows a high average because of noisy blobs and that the room is small and they are nearly always close to each other.

transition cost to disallow very costly links without affecting the decision making process. This will cause a substantial reduction in the number of links per node when people are far apart. If all the people stay far apart, which is not the common case, then the reduction could be up to having only one link because all merges and swaps are eliminated if the interstage time is short. Nodes that no other node links to can also be removed at this point. In the worst case, though, when people come close together, the number of nodes per stage and the number of links will again make the cost O(kb^{2t})) as described in section 3.6. The table 5.1 shows the decrease in the number of edges per stage that arise if only links are pruned based on distance. The reduction will be much higher if nodes not linked to the start node are also pruned.

Another reduction can be taken when the distance matrix is first calculated, and before possibilities are generated for the nodes. If a blob is very far from all blobs in the previous stage, it can be rendered invalid and not considered at all. This would take care of noisy or spurious blobs. In one of the test runs, for example, such a blob occurred from a monitor being on behind one of the screen walls.

Depending on the amount of merging expected in a run, this pruning and thresholding should drastically reduce complexity either until people come close together or if the time between two stages is very long. The implemented system can be run either with or without the edge thresholding.

5.3.1 Greedy Commits and Dynamic Window Size

Another variation using distance pruning can use a dynamically changing window size, and commits as many stages as it is "confident" about. The window size can then be changed based on the number of nodes generated by the newest stage.

First, when the system is started, all possibilities are computed from the start node, and if only one is found because of the distance pruning (all people far apart), then the algorithm could immediately commit that stage instead of waiting for the length of the window. This continues as long as only one possibility is linked. When more possibilities are created because the people come closer together, then the algorithm collects stages and searches the graph using DP at each iteration. It keeps adding stages as in chapter 3 until a certain threshold lowest cost from the start node is reached. Then, it behaves exactly as the processes described in chapter 3 to commit the new chosen node. When it is satisfied with the score, it commits the best node from the next stage.

In order for the threshold to accommodate changing window sizes and reflect the confidence of choosing a possibility over others, it should be a ratio of the two lowest cost from the start node. If the best node is, for example, an order of magnitude better than the next best, gathering a few more stages will most probably not change the path and the best next node may be committed. On the other hand, if the two are close, then more stages will provide a clearer indication as to which of the nodes is truly the best. This threshold ratio can thus be used as a decision confidence measure.

After the first commit, it tries to commit as many stages as it can, without searching the graph again, since the dynamic programming search has saved best distances to the end in all the nodes. When a stage is reached where a decision is unclear(the two best nodes' ratio is close to one, for example), more stages are gathered, up to a maximum stage limit set by the user.

The benefit of this scheme is that it will reduce waiting time for a decision if one is much better than its siblings, and reserve the bulk of the computation for the trickier situations. This is especially useful when there is only one person in the space, or

people that stay far apart and a short interstage timing.

On the other hand, one may end up mistakenly pruning nodes that may turn out to be useful later. The node that has a low confidence may look much better after a few stages due to noise in the frame it was found in, for example. Also there will be minimal to no pruning if the inter-stage time is too long. This can be ameliorated by carefully setting the ratio threshold such that pruning is minimal, the maximum window size such that it doesn't keep adding stages, and the interstage timing so as not to lose the distance metric. However, this drawback may not be eliminated. The trade-off is one of speed over accuracy.

Two more detailed schemes for handling the scalability will be discussed in Chapter 7.

Chapter 6

Results

The algorithm was run on 30 minutes of videos taken in a 6mx4m room in the lab shown in figure 1-1, with hard wood floors and furniture consisting of a fold away bed, a table, and a few chairs. Videos were shot with up to 4 people walking around the space. The people were told to walk around randomly and to meet and talk to each other as they walked around. In some sequences, people were asked to wear a bright color. In most of them, however, people walked in with what they were wearing the day the video was shot.

There are eight sequences that vary in length between two to fifteen minutes. People generally met in all areas of the room multiple times, in groups of two, three, or four. People also sat on the floor or on the chairs in the room, jumped up and down, waved their arms about, and leaned backwards and forwards as they passed under the camera.

The algorithm was implemented using Java. On a 300MHz dual processor Pentium II, each iteration, with distance pruning for distance penalties above 8, takes between 200-250ms for 2 people, 200-350ms for 3 people, and 250-600ms for four people. The higher times are when all blobs are close together but not yet merged. These higher times are also the result for when no pruning was performed. Without pruning, the 4 people run was slightly above the 500ms inter-stage timing used and that made the distance metric less useful, affecting the tracking negatively.

6.1 Error Classification

When people are far apart, the algorithm does not mix them up due to the distance metric, and mix-ups may only occur after merges or when people pass very close to each other. A merge occurs when two or more people's blobs merge into a single blob because they are close together. The number of mistakes the tracker can make after a merge is proportional to the number of people, and so merges were counted based on the number of people in them. Therefore, a 2-person group is counted as two merges, a 3-person group as 3 merges, and so on. Errors are then counted based on how many people are mistakenly assigned after they leave the merge. These errors are marked as "post-merge-errors".

Two more errors are shown in figure 6-1, and counted in table 6.2. These depend on the number of frames in which people are in a group, because they arise from the segmentation of such a group. They only occur *during* merges, and not when people pass by each other without becoming one blob. One occurs when the segmentation is unreliable because people are close enough for the system to pick up different parts of different people as separate blobs. These errors are marked "merge segmentation errors"(MSE). During these errors it is unclear who should be assigned to which blob because a person may belong to more than one. Because the histograms are not updated in this case and the distances of all the blobs found near the merge are very close, the assignments will not hurt the models.

The second error occurred when people were still in a group and the algorithm picked up a few overlapping blobs in which one corresponded to a complete person from the group, and that person was mislabeled. This error is marked as a "group proximity error"(GPE). If a person leaves the group and is still mistakenly assigned, then a "post merge error" is also counted.

A post-merge error is counted once for each person who merges with an assignment, and leaves the merged blob with another assignment. The algorithm will not be able to correct for that decision if it is actually committed and the person moves away. However, because of the nine histograms per person for the different areas of

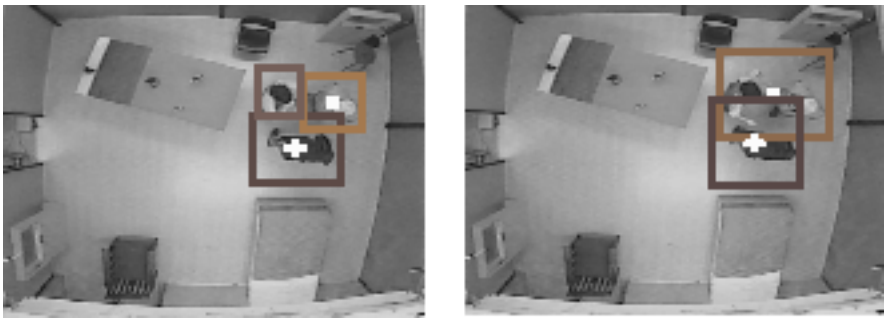

Figure 6-1: Group Proximity Error(GPE) and Merge Segmentation Error(MSE). These errors occurred only during the time people are in a group, and not when they pass by each other. Left: GPE: the person in the top right corner is properly segmented, but has been assigned as merged with the person below her. Right: MSE: the blobs do not correspond properly to the people. The system knows all the people are in that merge somewhere, but exact assignment is unclear.

the room, if two people get swapped and then come close together in an area where histograms have not been updated with the mistake, they may regain their initial assignments.

6.2 Results

The algorithm was tested with a number of window sizes and inter-frame timings, and the one that gave good results without causing too much delay was a window size of 5, with 500ms between stages. This causes a 2.5 second delay between the time a frame is taken and a decision on it is committed. A decision is made at 2Hz. Errors and merges were counted as described in section 6.1. Here, the post-merge errors are counted for runs on the same sequences with window sizes 2, 5, and then 10. These correspond to reasoning over windows of 1, 2.5, and 5 seconds respectively. The results are the same with or without pruning, except for the four person run whose results are shown here using pruning on distance penalties larger than 8. This was done because without pruning each iteration takes 600ms, which increases the interstage timing. With pruning, this upper bound is reached on only a few stages, and the 500ms inter-stage timing is generally maintained.

The table also shows what occurs if instantaneous decisions are taken, and there

Sequences	Merges	Post-Merge Errors on Window Sizes 1/2/5/10
2 people	4	3/0/0/0
3 people	54	22/3/0/0
4 people	48	19/17/14/12

Table 6.1: Number of people mistakenly assigned after leaving a group is shown for the window sizes of 2,5, and 10. The number of people mistakenly assigned is also shown for a window of size 1, in which the algorithm commits on every frame by taking the minimum of the transition costs from the previous frame and there is no delay on decisions.

# frames in Group	MSE	GPE
632	45	17

Table 6.2: Errors caused when people are close together. The number of frames is the number of frames used as stages, over all the sequences, containing people close together. Merge Segmentation Errors(MSE) occur when the segmentation does not provide blobs that properly correspond to people. Group Proximity Errors occur when people are in a group and one of them is segmented properly, but mislabeled. These number where counted for a window of size 5, and interstage timing of 500ms.

is no multiple hypothesis reasoning over time. This is illustrated by the results of running on a window of size 1. In this case the algorithm commits a decision on every frame, for the newest frame. It commits the decision whose node has the minimum transition cost from the start node. This was done without putting a minimum threshold on the interstage timing, and ran at approximately 5Hz.

As can be seen, most mistaken assignments occurred when four people where in the room (12/48). This is due to the size of so many people with respect to the space provided, and to the temporal proximity of merging. Four people had little space to move independently in the room, and would often go from one merge to another very quickly without giving the tracker a chance to get a clear shot of people leaving the group. It is also because three of the people had parts of their clothing with similar colors.

As expected, the number of errors decreases as the window size is increased and the algorithm collects evidence for the best hypothesis over the temporal window.

The segmentation and proximity errors did not affect the tracking in the sense that the tracker would give a position that is only a few pixels off to one side and did

not touch the histograms of the models. These show the instability of the data when a group of people is interacting. The causes of these results are summarized in the limitations section below.

6.3 Limitations

From the results above, it was seen that the algorithm is most likely to misassign people in the following cases:

Bad segmentation: This caused a person to be cut up into multiple blobs, or caused people in a group to be segmented into blobs that don't necessarily contain a full distinct person each.

Similar colors: If people are wearing very similar colors and merge with each other, they are more likely to be swapped. Since histograms carry no spatial component, people may be dressed differently but still have similar colors. This was seen in the four person sequence were one person (person1) is wearing a navy and white shirt and navy pants, and another (person2) is wearing navy pants and a white jacket with a few black stripes. This attribute was also found in that sequence between person 2 and a person (person3) wearing a black shirt with a big white circle on the front and back. Although the tracker got most of the merges, person 1 and person 2, as well as person 2 and person 3 were the ones that caused confusion. The fourth person, in a terracotta t-shirt and jeans, was never mixed with anyone else when the window size was above 5, and was tracked properly throughout the run.

Very Close, Consecutive Merges: An example is if person A and person B are in a group and then person B goes directly to merge with person C. The algorithm might never get a properly segmented blob of person B leaving the first group, and what it does get might make it think over time that person A is the one who moved to the new group. If, on the other hand, person C is a little farther away, and the algorithm gets a couple of shots of person B walking over, then

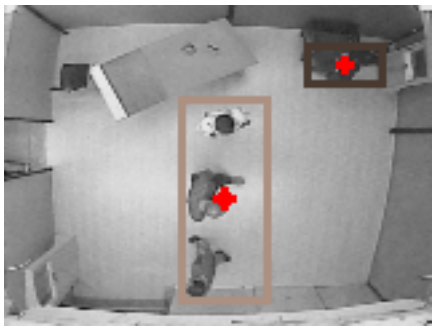

Figure 6-2: Three people side by side take up nearly the entire room. They are all in one blob, as can be seen by the single bounding box encompassing all of them as they walk past each other.

it does really well, as was seen in the sequences run. This problem with such quick, successive merges was only seen in one four person sequence. If people nearly fill up the space when they are standing relatively far apart, they will always be very close to each other. In this case, they may occasionally go from one merge into the other almost immediately. In the room used, four people standing, well-spaced, side by side will have blobs whose bounding boxes nearly span the length of the room. This is illustrated in figure 6-2 showing three people side by side in one blob that spans the width of the room.

Complexity: The algorithm as it stands was run with up to four people. Pruning as discussed in section 5.3 and using multiple graphs as in section 7 will decrease the complexity for applications that require more people.

6.4 Parameter Dependency: Setting Graph Size and Spacing

A number of parameters may be changed in the algorithm. Of these, the merge cost, window spacing and inter-frame time are the ones that have the most effect on running time and performance.

Although the tracker prefers a long window size so it can reason over time in cases of uncertainty, the choice of window size and inter-frame distance are heavily

dependent on the time one is willing to wait for a decision, the processing power available, and the required accuracy.

The tracker does best if it gets well segmented blobs around a merge, because the segmentation there can be unstable as people approach and separate. There are two options available for doing this: one is not taking a stage when the data is unstable, and the other is having a window size that is long enough so it can get a good shot of the people after they split before making its decision. Both options are discussed below, as well as other effects of changing window size and inter-frame timing.

6.4.1 Merge Cost

The merge cost, M in equation 3.1, was chosen by comparing it to the values returned by the histogram and distance metrics and finding a value is not so large as to overshadow them completely in one transition and not so small that it takes a very large window to accumulate enough to be considered. That set it above 0.2 for this implementation. Then, values were tried to get the best behavior over the recorded sequences. The final value of 0.35 was used to get the results shown in tables 6.1.

Increasing the merge cost will create a tendency to assign a person who is in a merge to a noisy blob near the merge, or to a person passing nearby if the merge has been there long. Decreasing the merge cost will create a tendency to merge people if one passes very close to another, and the second person stays still. In this case, the distance metric will be so good that it will decide on having the two people in the second person's blob. The value of 0.35 gave the best trade-offs for these two extremes.

6.4.2 Window Size

A long window size increases complexity by a constant factor, and if the inter-frame time is not very short it may also delay decisions by longer than is acceptable. It follows from the discussion that the longer the window size, the more temporal continuity can be propagated. This means that the decisions will get better. On the other

hand, it also means that a current decision may be completely changed because of a noisy frame ten stages ahead. The window size which gave good performance on the implemented tracker was 5. The performance deteriorated grossly when that was decreased to 2.

This value, though, is heavily dependent on the inter-stage timing. If that timing is small, then a shorter window will do better when people are somewhat far apart and the distance is the main metric because people will not be able to move much. On the other hand, it will need to be longer so that it can get a clear shot of people leaving a split before it commits on the noisy frames in between. These trade-offs will be discussed below.

6.4.3 Inter-stage Timing

The inter-stage timing can either be set to a constant or change based on the incoming data. First, consider the constant timing. In this case, a short time will give better distance based matching, but will introduce more noisy frames that occur around merges and splits as discussed above. It will also require a longer window size. It is limited by the available processing power, so it must be more than the time it takes to create a new stage, link it, search the graph, and commit a decision. This will depend again on the size of the graph. An inter-stage time that is too long will render the distance metric useless as it will drop down to zero, and put the bulk of the decision-making on the histogram score. This score in itself is not absolutely reliable, especially since people are non-rigid and their histograms may change if they lie down, bend to pick up something, or if they are wearing pants of a different color that only show from above when they take big steps. The histogram score does not include any information about the spatial relations of the colors. Therefore, depending on that metric alone will lead to error.

In light of the above, the timing chosen for the implemented system was 500ms, in conjunction with a window size of 5. At about 1000ms, the performance took a sizeable hit as people across the space got swapped because continuity of motion measures were lost.

One may also take stages depending on the stability of the input. This was implemented by manually telling the system when to take a new stage. The timing was such that it was frequent when people were far apart, and if they merged, would take the new stage when they became one clear blob. This would skip over the noisy segmentation frames as people come close together. Then, stages were taken of the people in the merge, and when they split there was a little wait until the segmentation stabilized and a clear shot of the people leaving could be taken. This improved performance because the tracker was not forced to make an assignment on some of the noisier frames. One may be able to automate this process by creating a stability measure, perhaps depending on the number of blobs in one small region, blobs enclosed in other blobs, and/or the ratio of blobs to people.

Chapter 7

Extending the Algorithm

A number of approaches can be taken to extend the algorithm so that it adapts to a particular space, uses the graph more efficiently, and handles special situations. A few of these will be discussed to show what possible steps could be taken next.

7.1 Training Weights, Tweaking Variables

The merge cost weight was chosen as described in section 6.4.1. The distance and histogram metrics are both unweighted. One may give them a weight and train the three numbers to get optimal performance for a specific location and frequency of merges. Training can be done either locally or globally. Local training involves getting the lowest value for the transition, at each stage, which corresponds to the correct assignment for that stage. The correct assignments can be hand coded, and then the tracker can update the weights to get the closest behavior. However, this approach ignores the time consistency of the tracker. Global training means changing the weights based on the number of mistakes the algorithm makes over a set of sequences for a particular choice of weights. This approach is better than local training for the general behavior of the algorithm, but takes a much longer time to train because each change in the weights requires rerunning, in our case, 30 minutes of video.

The weights are interdependent, causing a choice of a weight update function to be difficult. One possibility lies in the direction of classifying the errors in a run into

classifications that depend on the errors, and use that to map back into finding out which weight to change, and whether to increase of decrease it. This classification should depend on the effects that increasing or decreasing each weight will have. For example, errors from swapping two people when they are relatively far apart implies increasing the distance penalty. Errors from assigning a person to a noisy blob near a merge imply decreasing the merge cost, while those of putting people in a merge when they should not be imply decreasing the merge cost.

Spaces with high activity levels, play spaces for example, can be accommodated with a higher allowable velocity. The time sensitive distance function can also be tweaked to either allow or disallow various activity levels.

The size of the grid the room is split into for the histograms can also be changed based on the room setup. Histogram updates can be meshed with the saved histograms instead of completely overwriting them.

7.2 Adding Features and Constraints

One can add new features in the transition cost calculation, such as size, a face recognition measure from a separate module and camera, a merged blob separation mechanism such as that used in W4 [9] which searches for specific body parts. The feature's measurement, though, can only depend on the previous stage, otherwise it will violate the structure of the graph. A different segmentation algorithm may, for example, detect heads in a blob and use that to measure the possibility of actually having a merge.

Size was not used in this implementation because the algorithm does not segment body parts, so a person with his arms out will suddenly look about double the size of when he had his arms by his side. The top down view also does not allow for size because people lying down also look much bigger than when they are standing upright. Thus, features to be added can be based on the camera position used and the segmentation algorithm that picks out the people.

Constraints can be added on the new features, and the result can be included in the

transition scores the same way the distance and histogram were used to incorporate constraints on motion and appearance.

Another possibility is adding this dynamic programming approach as a layer above an existing tracking system to get temporal consistency.

7.3 More People, Complexity, and the Graph

In addition to the distance based edge-thresholding one can use on the links as described in section 5.3, one can also spend time on which nodes to link based on knowledge about what has happened. This may be done when new cases are to be handled, such as people entering and leaving the room. A scheme for handling entrances and exits will be discussed in Section 7.3.2.

Decreasing the number of nodes based on a distance threshold, though , still gives the worst case performance once people come close together. The tracker get slower as the number of people in the space increases. The tracker was run with up to four people, giving at worst 2^{16} edges per stage if there are no noisy blobs. This ran at half the speed of when three people where in the room.

7.3.1 Handling More Objects and People

In order to handle more people or objects, one may split the system such that it starts up more than one graph that get built simultaneously and handle a certain number of the tracked objects each. This would drastically reduce the complexity. For example, with six people and two trackers handling 3 people each, the complexity goes down from $\theta(k6^{12}) = \theta(k2^{36})$ to $\theta(k6^{6})) = \theta(k2^{12})$.

Although this reduction is favorable, the tracking will take a performance hit because the people are not independent. In order for this to be a realizable solution, one must take a number of steps.

First, the stages for both graphs must be taken simultaneously. When a decision is to be committed, the tracker must look at the decision from both and check for merges that result from the first graph and the second half putting someone in the

same blob. Also, the merge cost cannot be used because the two graphs must be independent and one cannot know how the cost would be affected based on the other graph's usage of the same blobs.

Second, the choice of which objects should be tracked by which graph is of crucial importance. People (or objects) that look similar should be tracked by the same half, because they are the ones that get swapped most often and are therefore the least independent of each other. A possibility is to split them by appearance, and another is to split them by function. For example, if one wants to track inanimate objects as well as people, one may separate the graphs such that one tracks the inanimate objects and one tracks the people and a heuristic for telling the inanimates from the people is used.

Most importantly, this approach will take a performance hit as less people are in the same graph because one is then ignoring dependencies between people's trajectories, histograms when they are merged, and appearance in the room. This will easily mix up two people that are dressed slightly similarly.

7.3.2 A scheme for Handling Exits and Entrances

People entering and exiting the space will have an effect on the graph, and must be specifically handled. When a person enters, a new model has to be created. When a person exits, the system has to generate nodes hypothesizing that any of the people may have exited until it builds up enough evidence to decide who it was that did so.

This section presents a scheme, that is currently in the middle of implementation, for handling these events, provided that people enter or exit one at a time. In an indoor space such as the room in which this system was implemented, people can only enter and exit through the door, so a module that monitors the door for activity was created. This module reliably signals whether someone has just entered or exited a room, and is described in detail in Appendix A.

In order to clearly describe this setup, an understanding of how models are used is important. In the current implementation, there is a certain number of available maximum models that correspond to the maximum number of people that are allowed

in the space at any one time. When a node state is generated, the extra models are simply marked with -1, while the valid ones are marked with which blob the hypothesis matches them to.

Entering

Upon getting a signal from the door module that someone has entered, the algorithm starts a new person model, with no histogram, and with initial position being the center of the door.

When the next new stage is about to be generated, the DP module will have one more valid model than it did when it generated the previous stage. At this point, it generates all possibilities as usual, and when it has to measure the distances from the previous stage for the new person model, it measures it as the distance from the center of the door to the blobs in the new frame. This is instead of the usual measure of what blob that person had been assigned to in the previous step, which for this new person would have been the invalid blob -1. The person will have no histogram either, and will be assigned a neutral score of 0.5 for her intersection with any of the new blobs. Then, the system continues as usual, and when the entrance stage is committed, the new person will have an assigned blob.

Exiting

This process is more complicated than the entering, because the algorithm has to figure out which of the person models to delete. The idea is that when an exit signal is received, the algorithm hypothesizes that each person, in turn, has left. Then it will keep generating these hypotheses until the exit stage reaches the beginning of the graph and is committed. At this point, it would have built up evidence for who it was that left, delete that person's model, and stop hypothesizing about exits until the next exit signal is received. However, the process gets trickier when one considers multiple exits within the same window, and entrances interleaved with exits. Also, one wants to minimize the nodes generated by the graph.

First, when the DP module receives a signal that someone has exited, it marks

the incoming stage as an "exit stage", and generates all possibilities such that each person, in turn, is hypothesized as exited. Exits are marked explicitly in the node state by assigning a person to -2. These new nodes are added to the end of the graph, and the distance to them is the distance from the blob the person was assigned to in the previous stage to the door. Histogram scores are again 0.5.

An exit count is incremented to signal that an exit stage is active. From that stage on and as long as the exit count is greater than zero, all new stages are generated in the manner above such that one person is hypothesized as exited. When they are linked to the graph, they must not provide a contradictory path that goes through one person exiting at stage t-1 and being valid again at stage t, or vice versa. Therefore, all transitions that go from any node that has a person marked exited (-2 instead of blob assignment) to any node in which that person is matched to a blob are disallowed and the two nodes do not get linked. This is if the new stage is not an "entrance stage". Similarly, all transitions from nodes in which a person is assigned to a blob to those in which the same person is marked exited are also disallowed, as long as the new stage is not a new exit stage. Links going from saying a person is exited to saying they are invalid are also not allowed unless the start node has just committed that person as having left and freed up his model.

The DP module continues searching the graph and committing nodes as it usually does. Then, when the exit stage reaches the start node and is about to be committed, the person who is committed as exited has her model set to invalid and is therefore no longer tracked. At this point the exit count is dropped by one, and the "search-commit-add-stage" cycle is resumed as long as no other exits are active. All nodes in the final stage who have that person marked exited then treat the exit assignment as if it were invalid when they are being linked to a new stage. All nodes with other people marked as exited are not linked to the new stage and will be unreachable from the start node because of the constraint on the links described above.

Multiple Exits in Same Window

If an exit stage occurs before a previous one is committed, the exit count is

again increased by one, and on the next round one more person is hypothesized as exited. The constraints in the links are the same as mentioned earlier, except now one person may go from being valid to being exited per link. The algorithm then proceeds as above.

Entrance Before Exit is Committed

An entrance stage may also occur while an exit stage is still active. An exit stage is still active if it has not yet been committed. At this point, the new person may start up any of the available, invalid people model. It may use the model of the person who has just left if that person's exit has just been committed. Therefore, the only case in which a link is allowed from a node in which a person is exited to one where that person is valid is if the new stage is an exit stage, and the start node has that same person marked as invalid. In this case, the transition cost treats it as going from an invalid to a valid state as described in the entering section above.

This scheme describes how people models can be reliably added and deleted from the system based on temporal consistency of decisions matching a blob to a new person and deleting a person's old model. The setup uses the constraint that people only enter and exit from the door, and works with a door detector module that is detailed in Appendix A.

7.4 Conclusion

The algorithm presented in this paper provides a methodology for tracking from visual input such that continuity over time is used to enforce constraints about movement and appearance. It provides an approach to handling low confidence decisions while tracking by gathering information over time to find which decision is supported by the subsequent steps. The approach can also be used to augment other tracking algorithms by providing temporal multiple hypothesis reasoning. A set of variations for handling complexity, trade-offs on allowable wait time for a decision, and han-

dling special situations such as the door are presented and discussed in detail. The algorithm was implemented in a room inside the House_n lab at MIT, and was tested with up to four people walking around the space. The best results were found using a window of size 5 with an interstage timing of 500ms. The delay between when a frame was taken and when a decision was made on it was therefore 2.5 seconds. A decision was taken at the rate of 2Hz. The algorithm did very well, although it could get confused if there is very bad segmentation, people are dressed similarly, or it cannot get a clear shot of them between merges.

Bibliography

[1] James Arnold, Scott Shaw, and Henry Pasternack. Efficient target tracking using dynamic programming. *IEEE Transactions on Aerospace and Electronic Systems*, 29(1), January 1993.

[2] David Beymer and Kurt Konolige. Real-time tracking of mutliple people using continuous detection. *http://www.ai.sri.com/ konolige/tracking.pdf*, 1999.

[3] Aaron F. Bobick, Stephen S. Intille, James W. Davis, Freedom Baird, and Claudio Pinhanez. The kidsroom: A perceptually-based interactive and immersive story environment. *PRESENCE: Teleoperators and Virtual Environments*, 8(4):387–391, August 1999.

[4] U.S. Census. Current population reports. *Statistical Abstract of the United States*, 2000.

[5] Leon Cooper and Mary W Cooper. *Introduction to Dynamic Programming*. Pergamon Press, 1981.

[6] T. Darrell, D. Demirdjian, N. Checka, and P. Felzenswalb. Plan-view trajectory estimation with dense stereo background models. In *MIT AI memos*, 2001.

[7] Davi Geiger, Alok Gupta, Luiz A. Costa, and John Vlontzos. Dynamic programming for detecting, tracking, and matching deformable contours. *IEEE Transactions on Pattern Analysis and MAchine Intelligence*, 17, March 1993.

[8] W.E.L Grimson, Chris Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proc. Computer Vision and Pattern Recognition Conference*, pages 22–29, 1998.

[9] Ismail Haritaoglu, David Harwood, and Larry S. Davis. W4: Real-time surveillance of people and their activities. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8), August 2000.

[10] Stephen S. Intille, James W. Davis, and Aaron F. Bobick. Real-time closed world tracking. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 697–703, June 1997.

[11] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision 29(1)*, pages 5–28, 1998.

[12] John Krumm, Brain Meyers, Barry Brumitt, Michael Hale, and Steve Shafer. Multi-camera multi-person tracking for easyliving. *Third IEEE International Workshop on Visual Surveillance*, July 2000.

[13] John MacCormick and Andrew Blake. A probabilistic exclusion principle for tracking multiple objects. In *Proc. Int. Conf. Computer Vision*, pages 572–578, 1999.

[14] John MacCormick and Michael Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. In *Proc.European Conf. Computer Vision*, volume 2, pages 3–19, 2000.

[15] David H. Nielsen, Donald G. Shurr, Jane C. Golden, and Kenneth Meier. Comparison of energy cost and gait efficiency during ambulation in below-knee amputees using different prosthetic feet - a preliminary report. http://www.oandp.org/jpo/11/1124.asp. *Journal of Prosthetics and Orthotics*, 1(1):24–31, 1989.

[16] T. Olson and F. Brill. Moving object detection and event recognition algorithms for smart cameras. In *Proc. DARPA Image Understanding Workshop*, pages 159–175, 1997.

[17] C. Rasmussen and G. Hager. Joint probabilistic techniques for tracking multi-part objects. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 16– 21, 1998.

[18] David K. Smith. *Dynamic Programming, a practical introduction*, chapter 2. Ellis Horwood, 1991.

[19] M. J. Swain and D. H. Ballard. Indexing via color histograms. In *Proc. IEEE Third International Conference on Computer Vision*, pages 390–393, 1990.

[20] M. J. Swain and D. H. Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.

[21] Kentaro Toyama, John Krumm, Barry Brumitt, and Brian Meyers. Wallflower: Principles and practice of background maintenance. *Internation Conference on Computer Vision*, pages 255–261, September 1999.

[22] C. R. Walters. A comparison of dynamic programming tracking methods. *Alogirthms for Tracking, IEE Colloquium on*, pages 9–13, 1995.

[23] Christopher Wren, Ali Azarbayejani, and Trevor Darrell. Pfinder: Real-time tracking of the human body. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(7), July 1997.

Appendix A

Door Detector

A person entering or leaving the room could be reliably detected visually using the door detector described in this appendix. This detector was implemented and tested on people walking in and out of the room used in this work. The idea for it came from infra-red emmiter receiver pairs that are set on opposite sides of a door frame and trip an alarm if the beam between them is interrupted. In order to tell an entrance from an exit,however, two such beams should be used.

Using the camera these beams were simulated by specifying two lines on the image, as shown in red in the left side of figure A-1. There is an outer line near the door on the outside of the room, and another line on the inside. These lines are both 2 pixels wide.

The detector uses the blobs found by the tracker and checks if any intersect the lines. Alternatively, one can get a background model for the pixels along the two lines in the same way as the tracker models the room, and then do the same background subtraction, in every frame, on the incoming pixels along these lines. The first method is chosen here since that information is already made available by the tracker. A line is considered "crossed" if more than a threshold number of its pixels are marked by the background subtraction as parts of blobs.

An entrance is signaled if only the outer line is crossed, then only the inner line is crossed within a specified amount of time. This signals a transition from the outside of the room to the inside of the room.

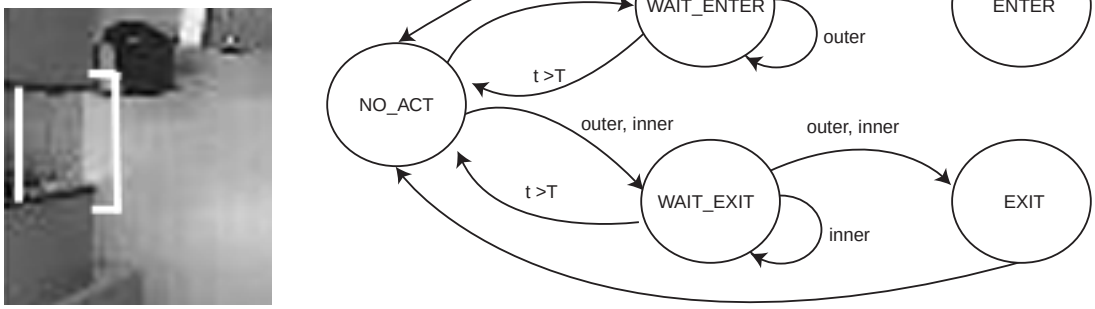

Figure A-1: LEFT: Lines around the door: Outer line is to the left, outside the room. Inner line is inside the room. RIGHT: Door Detector Finite State Machine: NO_ACT signifies no activity in the doorway. If the outer line is crossed, and not the inner one, then it waits for a threshold time for an entrance in the WAIT_ENTER state. The entrance is found if it then the inner line is crossed without the outer one. The opposite is done for the exit.

An exit is signaled if only the inner line is crossed, then only the outer line is crossed, within a specified amount of time. This signals a transition from the outside to the inside of a room.

The detector was implemented as a finite state machine(FSM), as illustrated in the right side of figure A-1. The *outer* means the outer line has been crossed, while "not outer" means it isn't crossed. The line being crossed is shown as *inner*, while "not inner" is that it isn't crossed. The algorithm waits 200ms after the first line is crossed for the second line to be crossed as well, as shown by the stages marked "WAIT". This is to give the person time to walk through the doorway.

In checking for these transitions, the detector avoids false-positives that could arise from people walking past the door, or stopping in the doorway but not going through it.