Design of a Person Tracking Algorithm for the Intelligent Room

by

Gregory Andrew Klanderman

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

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1995

Signature of Author .....
Department of Electrical Engineering and Computer Science
August 18, 1995

Certified by .....
W. Eric L. Grimson
Professor of Computer Science
Thesis Supervisor

Accepted by .....
F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Design of a Person Tracking Algorithm for the Intelligent Room

by

Gregory Andrew Klanderman

Submitted to the Department of Electrical Engineering and Computer Science
on August 18, 1995, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In order to build a room which can monitor the activities of its occupants and assist them naturally in the tasks they are performing, it is first necessary to be able to determine the locations and identities of its occupants. In this thesis, we analyze several algorithms proposed for visually tracking people interacting in such a computer-monitored "intelligent room". Through both experiments on real data and theoretical analysis of the algorithms, we will characterize the strengths and weaknesses of these algorithms. Given this analysis, we will discuss the implications for building a person tracking system using some combination these algorithms. We finish by proposing further experiments and an algorithm to solve the person tracking problem. This algorithm will form the basis for higher-level visual identification and gesture recognition for the Intelligent Room currently being built at the MIT Artificial Intelligence Laboratory.

Thesis Supervisor: W. Eric L. Grimson
Title: Professor of Computer Science

Acknowledgments

I would like to thank my advisor Eric Grimson for all the direction and support he has provided which were essential to the completion of this work. I would also like to thank Tomás Lozano-Pérez for his direction while Eric was on sabbatical and for pushing me to work hard in the early stages of this work. In addition, I have benefited greatly from many conversations with the members of both the HCI and WELG groups. My officemates Pamela Lipson and Aparna Lakshmi Ratan certainly deserve many thanks for all their help and for putting up with me during the last three years. And of course, Dan Huttenlocher cannot be thanked enough for getting me interested in computer vision and for always looking out for me. Special thanks also go to Sajit Rao, Miguel Schneider, Jae Noh, and William Rucklidge who generously provided code which I used extensively. My many friends at the AI Lab, MIT, and elsewhere must also be thanked. Their humor and the many diversions they provided somehow kept me sane throughout this endeavor. Finally and most importantly, I would like to thank my family without whose constant support throughout my entire life I could never have gotten here.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology, and was funded in part by the Advanced Research Projects Agency of the Department of Defense under Image Understanding contract number N00014-94-01-0994 and the Air Force Office of Sponsored Research under contract number F49620-93-1-0604.

Contents

Chapter 1

Introduction

At the MIT Artificial Intelligence Laboratory, we have embarked upon building the prototype system for a new paradigm in human-computer interaction. The goal is to eliminate explicit interaction with the computer in favor of a natural, seamless interface with computing resources. This natural interface would make interacting with the computer much like interacting with another human. The interface will take the form of an intelligent room which is able to interpret and augment human activity within the room. With the computer acting as an observer and participant, people in the room will be able to transparently command a vast array of computational and communication resources without even thinking about it. Instead, they will concentrate on solving real problems.

The applications we envision for such a room revolve around collaborative planning, rehearsal, and technical design. For example, a remote doctor assists in a surgery or one or more doctors plan a surgery, rehearsing several scenarios. Augmented reality eyeglass displays show a segmented model of the patient's anatomy. Tactile feedback allows probing of anatomical structures. Hand gestures allow the surgeon to change viewpoint, and voice commands allow him to remove structures to see what is beneath. Intelligent software agents look for relevant information in the patient's history and search for similar cases the doctor has seen.

As another example, in the context of collaborative mechanical design, engineers view a model of the system being designed. Augmented reality glasses automatically highlight different aspects of the design based on the current discussion or background and interests of each participant. As the discussion progresses, 'minutes' are automatically kept containing who said what for future reference. The room keeps track of who has the floor and allows that person to manipulate the shared view and modify the design using natural hand gestures.

The technology needed to build such a room consists of various presentation devices (augmented reality glasses, large screen displays, tactile feedback), intelligent software agents, a perceptual system to allow natural interaction, and a coordination and control system. In this thesis, we are

interested in building a basic component of the room's perceptual system. The perceptual system has the task of observing the people in the room and understanding what they are doing. It has to locate the occupants, identify them, recognize large and small scale gestures they are making, estimate where they are looking, estimate where they are pointing, determine their facial expressions, and recognize who is leading the discussion (who 'has the floor'). We propose to implement this part of the perceptual functionality visually, using a number of active and passive video cameras placed in the room to observe the occupants. Of course, other non-visual perceptual components will be needed, such as a voice recognition system to allow voice interaction.

This research is concerned with developing a low level person tracking system on which to base the higher lever visual processes of the room. This system will be responsible for tracking the location of each of the room's occupants through time. Since many higher level processes will be using this information, it must be very robust. In this thesis, we present research on the design of such a person tracking system. We will evaluate the performance of three tracking systems on many video sequences to characterize their strengths and weaknesses. Given this analysis, we will discuss the implications for building the person tracking system for the Intelligent Room using some combination these algorithms. We finishing by proposing further experiments and an algorithm to solve the person tracking problem.

1.1 Person Tracking

In order visually to observe people with the goal of understanding their gestures and allowing natural interaction with the computer to take place, we must first know where the people are so we can have some idea of where to look for these gestures. For example, we propose to recognize pointing and other hand and arm gestures, to determine gaze direction and facial expression, and to use face recognition to identify the participants. Since the room is a large and unstructured space with people allowed to come and go, these tasks are much easier if we can constrain the search space by having a rough idea where to look for arms, hands, and faces as well as how many to expect. Also, if we can recognize who has the floor (the person leading the discussion), we can direct the computational resources to that person's needs.

Thus, we need a multiple person tracking system to localize each person in the room and track these positions over time. At each time, the system should estimate the pose and location of each person and determine the correspondences with previous estimates. We will refer to determining correspondences as solving the 'person constancy problem'. It is not enough simply to know where people are; we need to know that the person now located at some position is the same one we observed at another location previously. This allows a higher-level process to assist the same person through time as they move around within the room.

The system must operate in real time and be capable of tracking complex human interactions over very long time periods (ideally, an hour or more). It must be very robust because higher level visual processing will be based on its estimates. Full recovery of the shape of the people in the room is probably unnecessary, however[14, 22]. Even if possible, this would require many orders of magnitude more computation than we can afford. Instead, the position and shape of a participant will be represented retinotopically, as the set of camera pixels comprising the image of that participant. Given the rough localization of each person, higher level process can determine what additional information is necessary for their task.

Figure 1-1 demonstrates the desired behavior of the multiple person tracking system. The figure shows two images from a sequence taken while three occupants were moving around in the room. The time elapsed between the two frames is a few seconds. Rough localization of the people has been accomplished by placing a bounding box around each person's outline in each frame. Further, correspondences between the detected people are represented by each person being given the same label in both frames.

In this thesis we consider only the restricted case of a single, fixed view, wide angle camera observing an area where people are interacting. We are interested in eventually incorporating multiple, possibly active cameras in the room for tracking, but this is outside the scope of the current research. In this context, the tracking system will determine the locations of the people present in each video frame as well as correspondences between the tracked people in consecutive frames.

Why should this tracking be done visually? We could use a pressure sensitive floor or transponders worn by the occupants, but these would not be able to give a rough segmentation of the people in the scene, just a single reference point. Also, a pressure sensitive floor would not solve the person constancy problem, and would likely be confused by chairs being moved around. Further, we would like to avoid having to instrument the people in the room; anyone should be able to enter and participate. Our goal is to make the room as non-intrusive as possible. Those who care to may wear enhanced reality glasses, but not all applications would need this technology so we hesitate to require it. Also, other applications of the technology being developed such as analysis of pedestrian traffic would not allow instrumenting each participant. We would like the technology required to be able to be added to an existing room as easily as possible. Requiring the floor to be instrumented with an array of sensors would be a significant impediment to transferring the technology into existing rooms. Finally, the higher level systems seem best solved visually.

1.2 Difficulty

The context of the room provides many constraints on this problem. For example, the only moving things are people or images on computer screens. The screens are in fixed and known locations.

Figure 1-1: Example demonstrating the desired behavior of the multiple person tracking system.

People are usually standing or sitting, and normally are upright. The background is relatively stable and lighting is mostly constant temporally, except for lighting changes and shadows. Most lighting changes will be computer initiated.

However, there are many difficulties. Keeping in mind the expected use for collaborative design and planning sessions, we expect participants in the room will often be close together and not moving a lot. Segmentation will be difficult with people nearby or occluding each other. People are inherently non-rigid and have many degrees of motion; arms and legs can be moving in very different directions and at different speeds. People may sit or stand still for long periods, then quickly turn around or bend over. Tracking all of these motions can be difficult. The person constancy problem is very difficult; it involves matching many similar non-rigid objects and distinguishing between them over time. When a participant is showing slides, low-lighting conditions may greatly reduce the contrast. Finally, a person gesturing in front of the shared large screen display may be difficult to track if the image on the screen is also changing.

1.3 Related Work

Much research in computer vision has been done on tracking both feature points and objects. Types of objects tracked vary widely. Tracking systems make various assumptions about the type of object to be tracked, for example whether or not the object is or is not rigid, articulated, or polyhedral. Tracking systems may be based on 3d or 2d models or may not be model based at all.

The basics of feature point tracking seem to be relatively well understood[21]. Correlation based methods which minimize sum of squared differences with respect to translation seem to work quite well in the case of small inter-frame displacements (See [1], for example). The problem of selecting good feature windows to track is more difficult. Obviously it is important that the feature window being tracked correspond to a real point in the world, not a motion boundary or specularity on a glossy surface. Shi and Tomasi[21] developed a promising selection criterion and feature monitoring method to select good features and detect when the template approach has lost the feature. A translation only model of image motion is used when matching from frame to frame, while an affine model is used over longer time periods to monitor if the original feature is still being tracked. Nowlan[17] has used convolutional neural networks to track a hand through hundreds of video frames and identify whether it is open or closed. The tracking system was able to track a number of hands correctly through 99.3 percent of the frames using a correlation based template match.

Lowe [13] has shown good tracking results for articulated polyhedral objects with partial occlusion. He uses an integrated treatment of matching and measurement errors, combined with a best-first search. The method handles large frame to frame motions and can track models with many degrees of freedom in real time. This method is probably not applicable to our needs since it

requires a 3d model of a rigid, articulated object comprised of straight line-segment features.

Woodfill and Zabih describe real-time tracking of a single non-rigid object[25]. They use dense motion field estimates based on consecutive frames to propagate a 2d model from frame to frame. The model is a boolean retinotopic map. The camera need not be fixed. At each frame, they adjust the model towards motion or stereo boundaries. They show results of tracking people on a CM-2 parallel computer. Having implemented this algorithm, we are skeptical about propagating optical flow estimates over many frames. Because there is noise in the optical flow computation, these errors get magnified over many frames causing the model to diverge and fill with gaps.

Huttenlocher et al.[10] have developed a model based tracking system based on 2d shape matching using a Hausdorff distance metric which works well. This is one of the algorithms we will be considering and will be described in detail in Section 2.3.

Bascle[2] integrates snake-based contour tracking with region based motion analysis. The motion field within the tracked object is used to estimate an affine motion of the object. This estimate is used to update the B-spline model and guide the initialization of a snake with affine motion constraints for segmentation in the next frame. Next, the affine constraints on the snake are relaxed giving the new object segmentation. Because the snake seeks edge boundaries, it does a good job segmenting the object. However, it requires a good initialization because it only uses edge information. Optical flow does not estimate region boundaries well, as seen by the failures of the Woodfill and Zabih method. But by using the whole region, it does estimate region motion quite precisely. This method has been used to track cars on a highway. It is unclear whether people would move fast enough to break the assumptions of mostly affine motion in our application.

Koller, et al. are also interested in tracking cars on a highway[11]. By using an adaptive background model, the objects are extracted then adjusted toward high spatial and time gradients. They use a Kalman filter to estimate the affine motion parameters of each object. Since the motion is constrained to the highway and the scene geometry is known, they can determine a depth ordering of the scene and therefore reason about occlusions. Again, it is unclear whether we can make the assumption of affine motion. Also, the ground plane assumption may not be useful to us if there are chairs and tables in the room which could occlude people's feet from view, as we expect to be the case.

Work on tracking people for measuring pedestrian traffic and intrusion detection has been done by Shio[22] and by Makarov[15]. Shio argues against recovery of full 3d structure in favor of 2d translational motion, considering algorithm robustness against noise and changing imaging conditions more important than high fidelity. He uses a background subtraction to segment the people from the background. By spatially propagating the motion field and smoothing the motion vectors temporally over a few seconds, the motions of all parts of a single person converge to the same value. Motion regions are then split and merged based on motion direction and a simple proba-

bilistic model of object shape to give a segmentation of the people. Makarov obtained good results for intrusion detection using a simple background subtraction on non-thinned edges. He notes that lighting changes are the main cause of false alarms for intrusion detection systems. Because of the prevalence of shadows and reflections in indoor scenes, ordinary background subtraction methods do not perform well. He obtained good results for his algorithm, but tested robustness to lighting change using an artificial additive model which does not effect the image derivatives upon which the edge detector is based except due to saturation.

Maes et al.[14] have developed a system which allows a user to interact with virtual agents. A camera projects an image of the user on a large screen and cartoon-like agents are virtually projected giving a 'magic mirror' effect. The system uses computer vision techniques to segment the user's silhouette and analyze his gestures. Very simple techniques suffice, because of the controlled environment. The constant background and floor and controlled lighting allow simple 'blue screen' subtraction to segment the user. The orientation with respect to the screen is controlled, since you must look at the screen. There is no furniture, making it easy to estimate depth based on the intersection of the ground plane with the user's feet. This system is similar to ours, however we are interested in allowing multiple people to interact in much less constrained environments.

Finally, much speculation exists regarding the form human-computer interaction will take in the future. For example, the video by Sun Microsystems[16] gives their vision. Many aspects of this vision are similar to ours, including multi-media presentation support using visual gesture recognition, speech understanding, enhanced visualization, access to global information, and remote participation.

1.4 Approach

In order to design a multiple person tracking algorithm for the Intelligent Room, we will first analyze three algorithms developed for similar purposes. All are based on having a single fixed camera observing the area in which we wish to allow intelligent interaction with the computer. The first is a correlation based feature tracker, similar to those described in [1], [24], and [21]. The second uses motion information to segment the image into moving objects. This tracker uses some of the ideas discussed in [25] but is greatly simplified. The final algorithm is a model based system employing 2D shape matching using the Hausdorff distance. This algorithm was developed by Huttenlocher, et al. [10]. Our hope is that the shape matching which it uses will enable us to distinguish between multiple people and solve the person constancy problem for tracking several people.

The planned experiments consist of a number of difficult cases obtained by reasoning about the algorithms and also observing their behavior in many situations. For example, in the motion based algorithm, if the person moves too fast, the search radius in the optical flow step will be too small

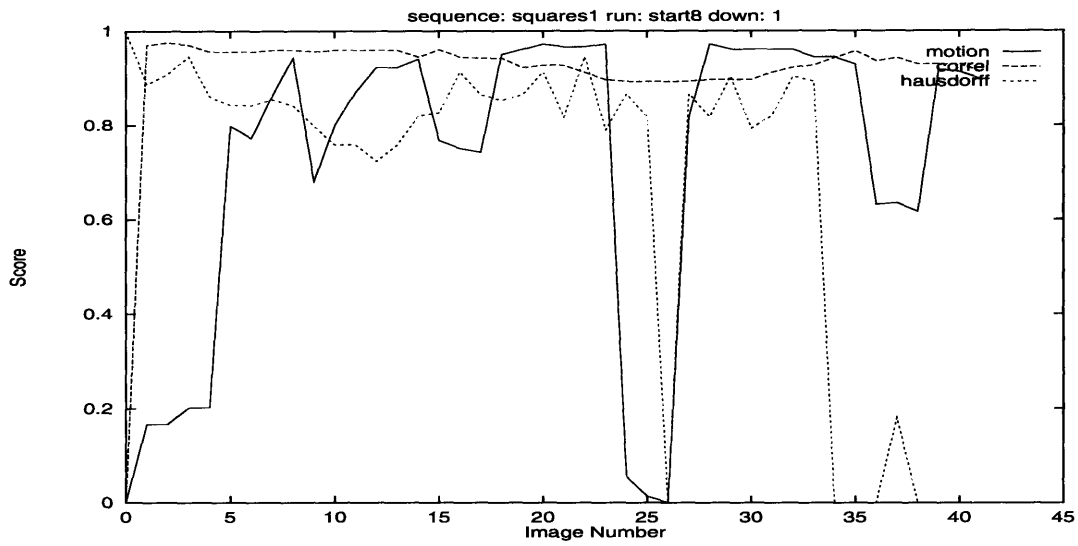


Figure 1-2: Example plot comparing the frame by frame scores of runs of the three algorithms on the full resolution `squares1` sequence.

and the algorithm should fail to find matches. The experiments involve a number of tests of the algorithms tracking simple objects making relatively simple motions as well as tracking real people interacting in the room.

Analysis of the experiments will involve scoring each algorithm's performance on each test video sequence. Ground truth data will allow us to score each algorithm's performance. Using this quantitative analysis as a guide, we can quickly analyze the strengths and failures of each algorithm qualitatively and uncover the reasons for each failure. Based on these failures, we then propose an algorithm that is less susceptible to these conditions.

Figure 1-2 shows an example of the type of quantitative analysis we will be using. Scores for each of the three algorithms we are studying are plotted as a function of image frame number for one of the video sequences we will be using. From the plot it is easy to see when the algorithms are performing well and when they are failing. We will be using this type of analysis extensively in Chapter 4.

The remainder of this thesis is organized as follows: Chapter 2 describes each of the algorithms we will be analyzing in detail. Chapter 3 describes the experimental design, including our image sequences, the experiments we will be doing, quantitative measurement of performance, and analysis. Chapter 4 describes each of our experiments and analyzes the results. Finally, Chapter 5 analyzes the implications of these experimental results for solving the person tracking problem and proposes further experiments and an algorithm to track people.

Chapter 2

Tracking Algorithms

As we have described previously, we will be analyzing three tracking algorithms and the features of which they make use. By understanding the conditions under which they succeed and fail, we will be able to suggest an algorithm to solve the multiple person tracking problem in the Intelligent Room. We have chosen these algorithms because the underlying features of the algorithms are representative of a wide range of tracking systems. The features used are correlation matching, difference images, optical flow (motion), and shape matching.

2.1 Correlation Based Feature Tracking Algorithm

The first algorithm we will consider is a correlation based feature point tracker. The basic algorithm is very simple and consists of tracking a small patch of an image of the object of interest from frame to frame, looking for the best match of the patch from the previous image in the current image. The particular implementation described was originally developed by Miguel Schneider[20] for an embedded TMS320C30 based vision system. Only minor changes were made in converting it to run on a UNIX platform.

Correlation based feature trackers are quite common for computer vision applications (see [1], [24], and [21]). They are also among the simplest tracking algorithms. The general framework for a correlation based feature tracker is shown in Figure 2-1. At each step, we first locate the feature being tracked in the current frame. Using the image patch surrounding the feature in the previous image as a template, we look in a region of the current image around the predicted new location for the best match with the template. This predicted location may be based on the previous trajectory, among other things. Using the best match location, we then update the template and trajectory information. Although the matching we use is based on minimizing a sum of squared differences, it is essentially equivalent to maximizing the correlation and hence we will refer to this algorithm as the correlation based feature tracker.

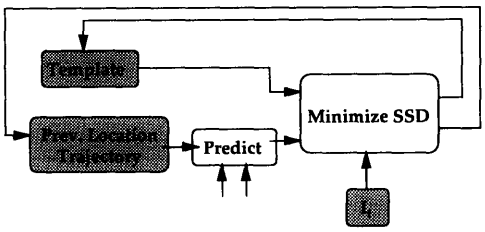


Figure 2-1: Correlation based feature-point tracker

Let $M_{t-1}[i,j]$ be the $N \times N$ feature template from time $t-1$. For convenience, let $N = 2R_P + 1$ where R_P is the patch radius and let the i and j indices of M_{t-1} range from $-R_P$ to R_P. This choice of coordinates allows the center of the patch to have coordinates $(0,0)$ and we designate this position the feature's location. Usually the patch is rather small in relation to the whole object being tracked and is centered on some local feature of that object. Given a new image $I_t[i,j]$ at time t, we search for the best match for the model patch M_{t-1} in the region of the image I_t of radius R_S surrounding the predicted position by minimizing the sum of square differences (SSD) measure. Given a predicted position of the feature in the current frame $(\hat{\imath}, \hat{\jmath})$ we compute the displacement to the best match from the prediction as

$$(i^*, j^*) = \arg \min_{-R_S \le i,j \le R_S} \mathrm{SSD}(\hat{\imath} + i, \hat{\jmath} + j). \tag{2.1}$$

The SSD measure is defined as

$$\mathrm{SSD}(i,j) = \sum_{k=-R_P}^{R_P} \sum_{l=-R_P}^{R_P} (M_{t-1}[k,l] - I_t[i+k, j+l])^2. \tag{2.2}$$

Once the displacement of the best match (i^*, j^*) has been determined, the point $(\hat{\imath} + i^*, \hat{\jmath} + j^*)$ is considered the new location of the feature, and this point is output. Finally, the model is updated to be the patch of the current image which matched best, and any trajectory information is updated.

Figure 2-2 shows an example of the correlation tracker tracking a person's head. The previous image, I_{t-1}, is shown in (a), with the bounding box of the model, M_{t-1}, outlined. A closeup of the model is shown in (b). The current image, I_t, is shown in (c). The SSD surface is shown in (d) for a predicted position of $(-24, 24)$ with respect to the position in the previous image. The center corresponds to $\mathrm{SSD}(0,0)$ and darker regions indicate smaller SSD values. The patch radius is 20. The minimum is at $(-9, -26)$ with a match SSD value of 374680 (about 15 grey values per patch pixel). This gives an overall displacement of $(-33, -2)$ with respect to the previous feature location. The new model M_t corresponding to this location is shown outlined in (c) and magnified in (e).

In our implementation, we use color images for the correlation matching if available. To compute the SSD in the RGB color space we simply sum the individual squared pixel differences from each

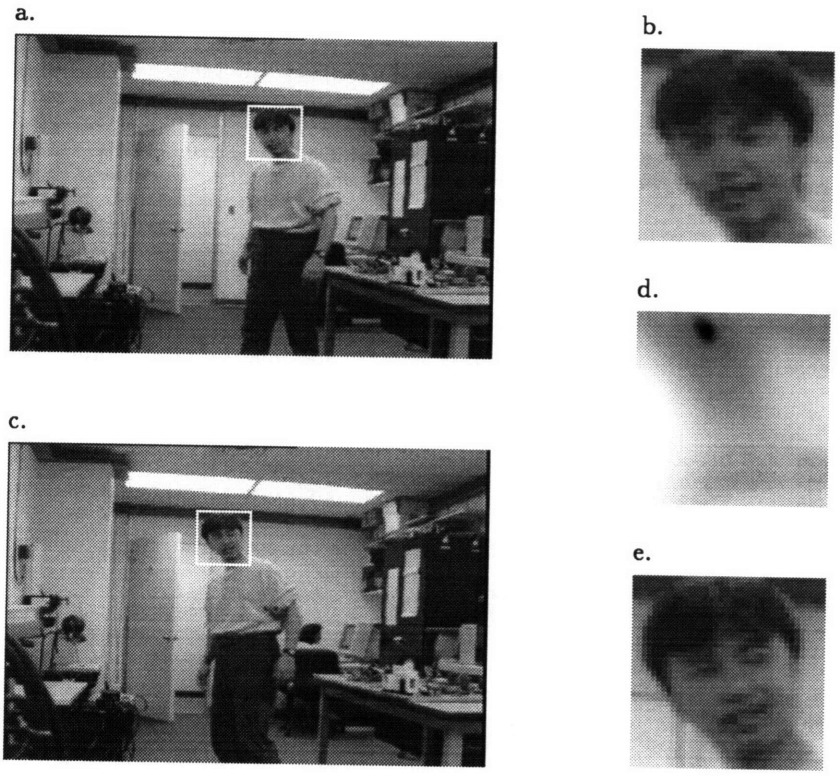


Figure 2-2: Correlation based tracker example: a. previous image showing the bounding box of the previous model feature patch, b. closeup of the previous model, c. current image, d. SSD surface, e. closeup of the new model patch

of the three channels. Also, our implementation actually makes two predictions for the new feature location. The first is centered at the previous detected location. The second is based on the "motion difference" within two windows centered on the previous detected feature location. The search for a match with the template is performed around each and the best matching location used.

To compute the motion difference successive images are first subtracted and the absolute value thresholded to give a binary difference image indicating where significant change has occurred. More precisely,

$$D_t[i,j] = \begin{cases} 1 & \text{if } |I_t[i,j] - I_{t-1}[i,j]| > \tau_D, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Then, successive difference images are compared to give the binary motion difference image. This image consists of those pixels where there is change in the current difference but not in the previous difference:

$$MD_t[i,j] = \begin{cases} 1 & \text{if } D_t[i,j] = 1 \text{ and } D_{t-1}[i,j] = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

Consider the can moving left to right in Figure 2-3. The corresponding difference images detect change at both the previous and new locations. However, the second difference, or motion difference image has almost entirely eliminated the response at the previous location while preserving the response at the new location.

The motion difference based prediction is computed from the first order moments of the motion difference image within two windows surrounding the previous feature location. The mean horizontal pixel index is computed within the horizontal window to give the estimated position $\hat{\imath}$ and the mean vertical pixel index is computed within the vertical window to give $\hat{\jmath}$:

$$\hat{\imath} = \frac{1}{N_H} \sum_{i=i^*-W_{HW}}^{i^*+W_{HW}} \sum_{j=j^*-H_{HW}}^{j^*+H_{HW}} i \cdot MD[i,j] \tag{2.5}$$

and

$$\hat{\jmath} = \frac{1}{N_V} \sum_{i=i^*-W_{VW}}^{i^*+W_{VW}} \sum_{j=j^*-H_{VW}}^{j^*+H_{VW}} j \cdot MD[i,j] \tag{2.6}$$

where (i^*, j^*) is the previous detected location, N_H and N_V are the number of motion difference pixels in the horizontal and vertical windows, and W_{HW}, W_{VW}, H_{HW}, and H_{VW} are the half widths and half heights of the horizontal and vertical windows.

An example of these windows on the can sequence is shown in Figure 2-4. The horizontal and vertical windows are shown centered on the previous patch location which is shown in light gray. The motion difference pixels are shown in black. The cross-hairs show the location of the predicted new location, which is very close to the true position shown by the small square patch.

Several additional prediction methods will be investigated and described with the other experi-

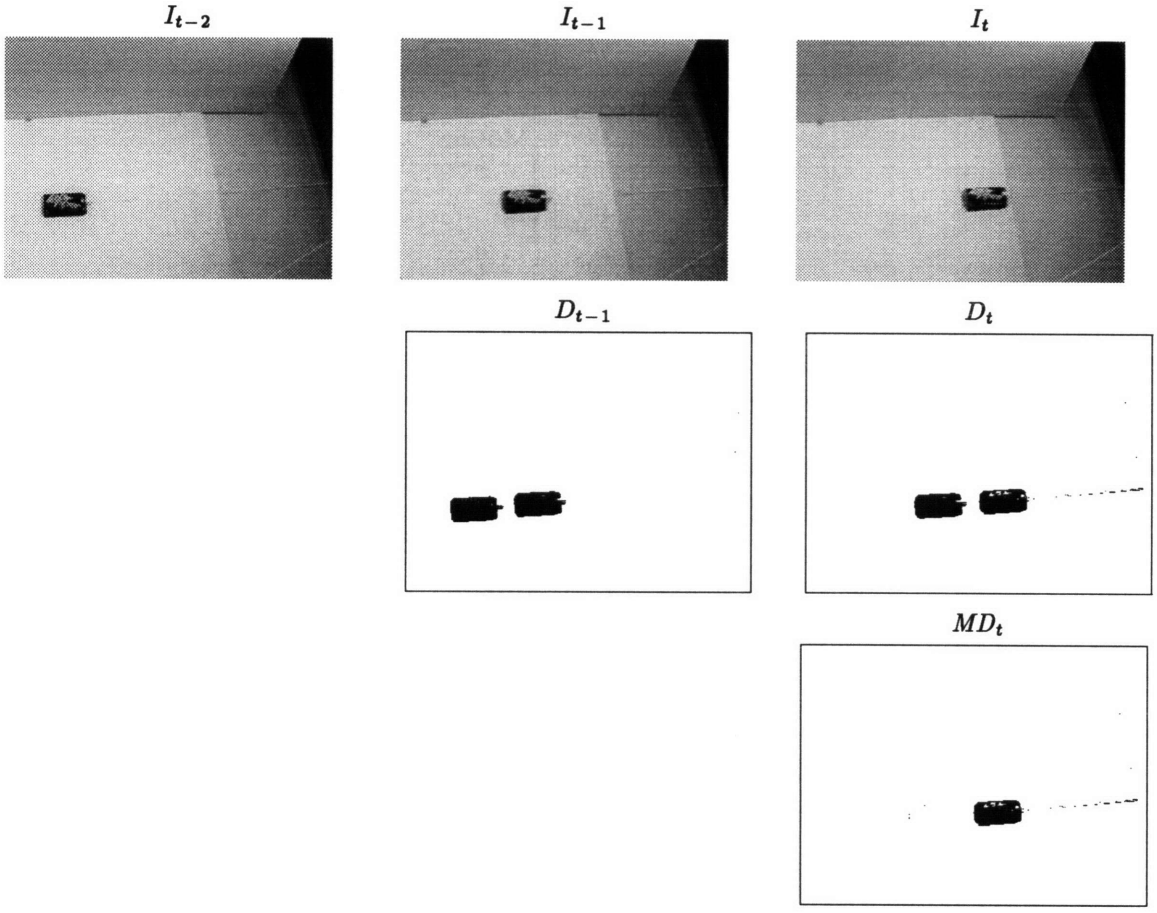


Figure 2-3: Motion difference computation. Top row: can moving left to right, center row: difference images, bottom row: motion difference.

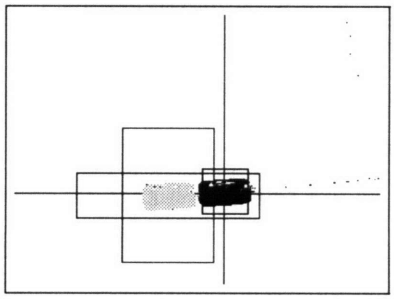


Figure 2-4: Computation of prediction based on motion difference windows

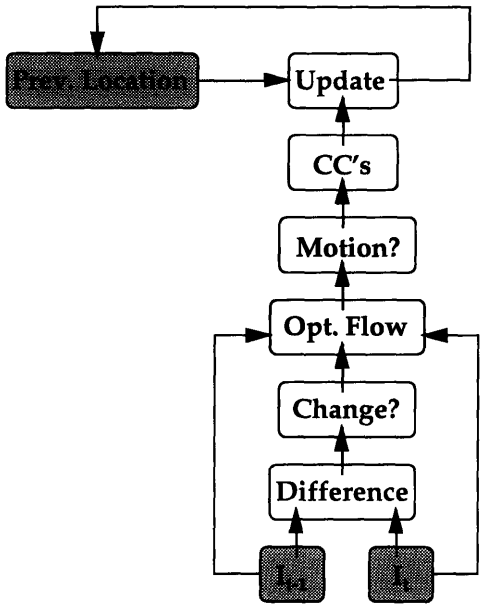


Figure 2-5: Motion based tracker

ments.

The parameter values we have used for this algorithm are as follows: The search radius for the first prediction, S_{R1}, is 2 and for the second (motion difference) prediction, S_{R2}, is 8. The patch size is 11 × 11 ($P_R = 5$). The horizontal window is 41 × 11 and the vertical window is 21 × 31 ($H_{HW} = 5, W_{HW} = 20, H_{VW} = 15, W_{VW} = 10$). All of these values are for 80 × 60 images and are scaled linearly with image size. The image difference threshold, τ_D, is 15 grey levels. These values of the parameters were found to give good results by Schneider and we have left them unchanged.

2.2 Motion Based Tracking Algorithm

The basic idea behind the second tracking method we will investigate is that the pixel locations in the image where there is motion can be grouped into connected regions corresponding to the objects we wish to track. The algorithm we will describe was developed by Satyajit Rao[19] in the summer of 1994 and is currently in use in the Intelligent Room for tracking single people at a time. Again, some minor modifications have been done to convert the algorithm to our platform, further instrument it, and investigate small variations.

A block diagram of this algorithm is shown in Figure 2-5. First the regions of the image where change has occurred since the previous image are identified. Optical flow is computed at these locations and the places where motion was detected are grouped into connected regions. Finally, these motion regions are put into correspondence with the previously detected regions.

The algorithm starts by computing the intensity difference between the current image I_t and

the previous image I_{t-1} to locate the pixels where significant change has taken place and focus the computation on these locations. These locations are most easily represented using a binary image where each pixel tells whether there was significant change in the corresponding location of the input image. This is identical to the difference image used for the motion difference based prediction in the correlation based feature tracker:

$$D_t[i,j] = \begin{cases} 1 & \text{if } |I_t[i,j] - I_{t-1}[i,j]| > \tau_D, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

Next, at each of the locations where there was significant change, eg. $D_t[i,j] = 1$, we compute the optical flow from the previous image to the current image. Optical flow gives a vector at each image pixel pointing to the location that pixel moves to in the next image. There are two common techniques for computing optical flow, those using correlation of small image patches between the two images and those using the constant brightness equation[8] on the spatial and time derivatives of image brightness. We use a correlation based algorithm similar to the feature point tracker described previously to get a dense set of correspondences between the previous and current images using small image patches. The matching is done using a sum of absolute differences (SAD) measure, similar to the SSD measure discussed previously. We denote by $\mathrm{SAD}(i,j,k,l)$ the match score of the square patch of radius R_P centered at (i,j) in image I_{t-1} with the patch centered at (k,l) in image I_t:

$$\mathrm{SAD}(i,j,k,l) = \sum_{m=-R_P}^{R_P} \sum_{n=-R_P}^{R_P} |M_{t-1}[i+m, j+n] - I_t[k+m, l+n]|. \tag{2.8}$$

The optical flow at a pixel (i,j) is computed as

$$\mathrm{OF}(i,j) = \begin{cases} \arg \min_{-R_S \le x,y \le R_S} \mathrm{SAD}(i,j,i+x,j+y) & \text{if } \min_{-R_S \le x,y \le R_S} \mathrm{SAD}(i,j,i+x,j+y) < \tau_M, \\ \langle \text{no-match} \rangle & \text{otherwise.} \end{cases} \tag{2.9}$$

This is simply the displacement within a small search window of radius R_S at which the patch centered at (i,j) in I_{t-1} best matches a patch in I_t. If the match score is not below the match threshold τ_M there is no good matching patch and the optical flow has the special value $\langle$no-match$\rangle$.

Finally, we can group the pixels where motion was detected into connected motion regions corresponding to moving objects. Given the optical flow values, we generate a binary image telling at each image pixel whether motion was detected there. We define the *MOTION* image:

$$MOTION[i,j] = \begin{cases} 1 & D_t[i,j] = 1 \text{ and } \mathrm{OF}(i,j) \neq \langle \text{no-match} \rangle \text{ and } \|\mathrm{OF}(i,j)\| > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

The grouping into regions can now be accomplished with a simple connected components algorithm

which labels any two pixels in the *MOTION* image within a distance of R_{CC} using the same label. Given these regions, all that remains is to determine correspondences between the objects being tracked and the motion regions. Since this algorithm was designed for tracking a single person, it uses a simple rule: it chooses the component closest to the last position the person was detected as the new location. If no motion was detected the person is considered to have stayed in the same place. The output consists of the bounding box for the motion region being tracked. Note that unlike the feature point tracker, this algorithm is able to track the whole moving object, not just a point on that object.

Examples of the steps of this algorithm can be seen in Figure 2-5. Images (a) and (b) show two consecutive frames of a sequence in which a cereal box is moving right to left. The difference image is shown in (c) and (d) shows the *MOTION* image containing the pixels where motion was detected. It is the same as the difference image, except for some boundary pixels where the optical flow is not computed due to the edge of the image. In (e), the two connected regions of motion pixels detected are shown, one in black and one in grey. Finally, in (f), the region corresponding to the cereal box which is being tracked has been selected and a bounding box and cross hairs drawn. The other region corresponds to a thin string the box is hanging by.

This algorithm is somewhat similar to that described by Woodfill and Zabih in [25]. However, each uses the motion information from the optical flow differently. Woodfill and Zabih propagate model from frame to frame using the detected flow, which tends to accumulate large errors over time. Rao, on the other hand, simply uses the flow to detect where motion is taking place based on two frames and then groups these locations. There is no accumulation of errors over long image sequences.

The values of the parameters we have used are: The patch radius for the optical flow, R_P, is 1 (3×3 patch) and the search radius, R_S, is 3 (7×7 region). The connected component radius, R_{CC}, is 4 pixels. The match threshold, τ_M, is 20 grey levels times the patch area in pixels. The difference threshold, τ_D, is 20 grey levels. The values for R_P, R_S, and R_{CC} given are for 80×60 images and scale linearly with image size.

As implemented, this algorithm uses only intensity information (grey images). It would be easy to extend both the differencing and optical flow steps to make use of multiple channels. The direction of the motion could also be used in the grouping stage to only group nearby pixels with the same motion direction, thereby possibly separating nearby moving objects better. We will not investigate how these changes would affect the performance of the algorithm.

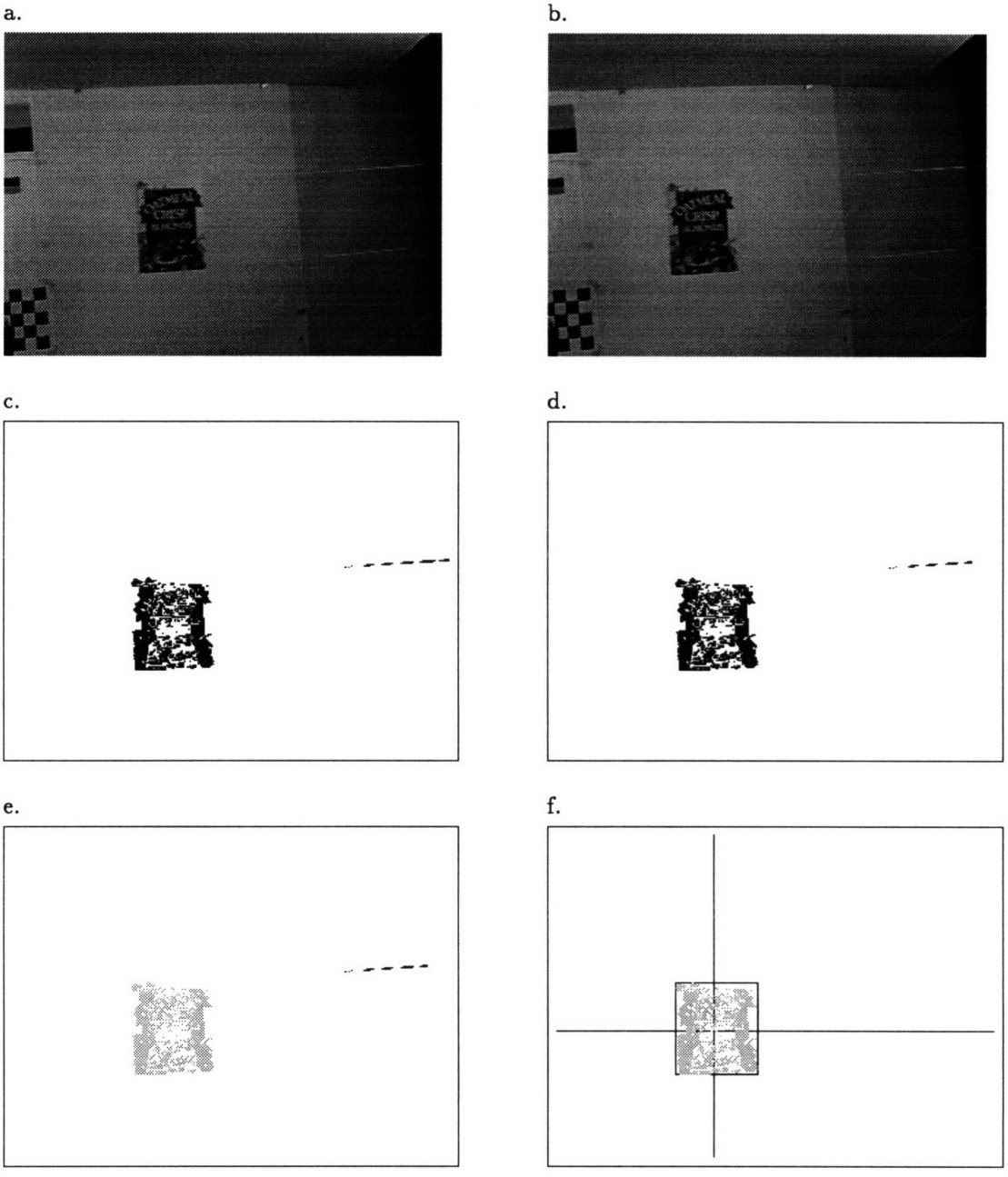


Figure 2-6: Motion based tracker example: a. previous image, b. current image, c. difference image, d. detected motion locations, e. two connected motion regions, f. new tracked motion region and bounding box

2.3 Hausdorff Distance Model Based Tracking Algorithm

The third and final algorithm we will be investigating is a shape based tracking system developed by Huttenlocher, et al.[10] and described there in detail. The method is based on a shape similarity metric, called the Hausdorff distance, applied to 2D image edge feature points. This metric has been shown to perform well in the presence of noise and partial occlusions and can be computed quickly[9]. It does not rely on establishing explicit correspondences between the image and model feature points but rather matches entire groups of edge points. The code we will be testing was obtained from the authors of [10]. Some work has been done to instrument it further and some minor modifications have been made.

The basic idea behind the algorithm is to match a 2D model of the shape of the object acquired from the previous image against the edges found in the current image, considering all possible locations. The best match location determines the position of the object in the new frame. Then, the model is updated to reflect the features matched in the current image. By exploiting the global shape of the object being tracked, the method should provide significant advantages over purely local methods when there is significant clutter in the environment, there are multiple moving objects, or there are large motions of the objects between frames. This is a major reason we have chosen this as one of our algorithms to test. We hope that the shape based approach will be able to distinguish between multiple people and solve the person constancy problem by matching the shapes of the people being tracked between frames.

The algorithm considers the change in the image of the object between frames to be composed of a 2D motion and a 2D shape change. This decomposition allows articulated or non-rigid objects to be tracked without requiring any model of the object's dynamics, so long as the shape change is small enough between frames that the shape matching metric does not fail. The shape matching will be based on the Hausdorff distance, which is inherently insensitive to small perturbations in the edge pixels comprising the shape of the object.

2.3.1 Comparing 2D Shapes Using the Hausdorff Distance

The Hausdorff distance measures the extent to which each point of a set A lies near some point in a set B and vice versa. It is defined as

$$H(A, B) = \max(h(A, B), h(B, A)) \tag{2.11}$$

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} ||a - b|| \tag{2.12}$$

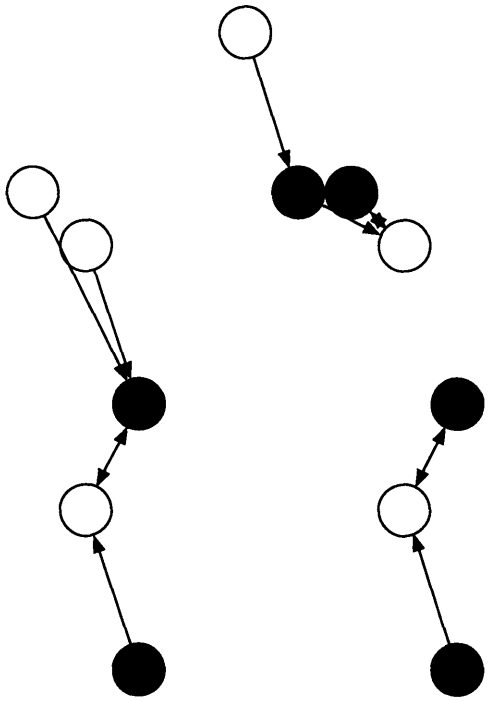


Figure 2-7: Example showing the computation of the Hausdorff distances between two point sets

and $||\cdot||$ is some underlying norm measuring the distance between the points of A and B. For our purposes we will use the Euclidean or L_2 norm. The function $h(A, B)$ is called the directed Hausdorff distance from A to B. It identifies the point $a \in A$ which is furthest from any point in B and gives the distance from a to the closest point in B. In effect, it computes the distance from each point in A to the closest point in B, ranks all these distances, and gives the maximal such distance. Hence, if $h(A, B) = d$ then each point of A is within a distance d of some point in B and there it at least one point in A which is exactly a distance d from the nearest point in B. Thus $h(A, B)$ tells how well the points in A match those in B.

The Hausdorff distance is the maximum of $h(A, B)$ and $h(B, A)$ and therefore measures the mismatch between the two point sets. If this distance is small, all points in A must be near points in B and vice versa. Figure 2-7 shows two point sets and illustrates the computation of the directed Hausdorff distances between them. The set A is represented by hollow circles and the set B by filled circles. Arrows from each point indicate the closest point in the other set. Hence, $h(A, B)$ is the length of the longest arrow from a hollow circle to a filled circle, and $H(A, B)$ is the length of the longest arrow.

The Hausdorff distance measures the mismatch between two point sets at fixed positions with respect to each other. By computing the minimum distance over all possible relative positions of the point sets we can determine the best relative positioning and how well the sets match at that positioning. The positions we consider can in general be any group of transformations. Here we will

consider only the set of translations of one set with respect to the other. Thus, we can define

$$M_T(A, B) = \min_t H(A \oplus t, B) \tag{2.13}$$

to be the minimum Hausdorff distance between A and B with respect to translation. In Figure 2-7, $H(A, B)$ may be large but $M_T(A, B)$ is small because there is a relative translation which makes the two point sets nearly identical.

We can use the Hausdorff distance to look for an instance of a model in an image. We consider the set A to be the 'model' and the set B to be the 'image' because it is most natural to consider the model translating with respect to the image. An edge detector or some other point feature detector can be used to get a set of points corresponding to the features of the model and image from intensity values. We then look for translations t for which $H(A \oplus t, B) < \delta$. We consider these translations of the model with respect to the image where the mismatch is small to be instances of the model. Note that because the full image may contain more than just an instance of the model, when performing the reverse (image to model) match, $h(B, A \oplus t)$, only those points in A which lie beneath the outline of the translated model should be considered. The distance threshold δ is set to account for noise in the edge detection process and to allow for small amounts of uncertainty in the model's expected appearance due to small amounts of rotation, scaling, shape change, etc.

One final modification to the Hausdorff distance is needed for this to work well in practice. The fact that the Hausdorff distance measures the most mismatched point makes it very sensitive to the presence of outlying points. In order to deal robustly with this situation and to allow the object of interest to be partly occluded, we introduce partial distances. In computing $h(A, B)$, we ranked the distances of points in A to the nearest point in B and chose the maximum distance. If instead, we choose the Kth ranked value we get a partial distance for K of the q model points:

$$h_K(A, B) = K^{th}_{a \in A} \min_{b \in B} ||a - b|| \tag{2.14}$$

where $K^{th}_{a \in A}$ denotes the Kth ranked value in the set of distances. Thus, if $h_K(A, B) = d$ then K of the model points are within distance d of image points. This definition automatically selects the K best matching points of A. In general, we choose some fraction f and set $K = \lfloor fq \rfloor$. It is easy to extend this idea to get a bidirectional partial distance, $H_{KL}(A, B)$.

Efficient rasterized algorithms have been developed for finding all translations of a model with respect to an image where the partial Hausdorff distance is below some threshold distance[9]. We will not go into the details of this here.

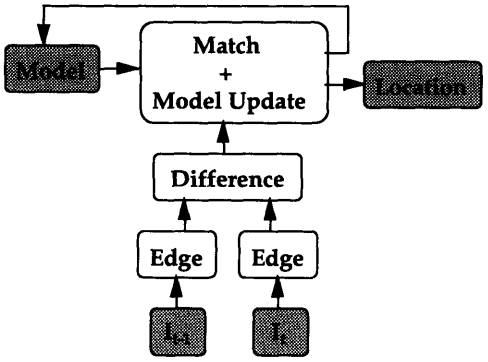


Figure 2-8: Hausdorff distance model based tracker

2.3.2 Model Based Tracking Using the Hausdorff Distance

A simplified block diagram for the model based tracker is shown in Figure 2-8. The fundamental part of the algorithm consists of two basic steps: locating the object in the current frame, and updating the object model. The inputs take the form of the current image, I_t, and the previous object model, M_{t-1}. A binary image representation is used, with the non-zero pixels representing sets of points. For simplicity, the model is a rectangular sub-image consisting of a subset of the pixels found in the image at the location the object was found.

The current image, I_t, is actually a processed version of the intensity image input from the camera. A method similar to [5] is first used to extract intensity edges from the input intensity image. Next, any edge pixels corresponding to stationary background which appeared in the output of the edge detector for the previous image are removed. Finally, a shot noise filter is applied to remove any remaining stray background pixels, yielding edge image I_t.

Locating the object in the current image consists of using the forward Hausdorff distance to locate the best match location of the previous model, M_{t-1}, in the current image I_t. The forward distance measures the extent to which some portion of the model resembles the image, so minimizing this distance over all possible relative translations gives the best match location. This minimum value of the forward distance is given by

$$d = \min_t h_K(M_{t-1} \oplus t, I_t)$$

and identifies the translation t^* minimizing Equation 2.14. Note that all possible translations are considered, not just those within some local search window. At this position, at least K of the translated model points are within a distance d of image points. This match location t^* is considered the new position of the object in the current frame and the bounding box of the model at that translation is output as the object's location.

When locating the model, a few other cases can arise. If there is more than one good match with

the current image, trajectory information is used to disambiguate and determine the most likely match. Objects are assumed to continue in the same direction they have been moving. If there is no good match (e.g. $d > \delta_M$ where δ_M is some match threshold), the tracker tries to locate a good match for a previous model, by searching a database of canonical models. The canonical models are a subset of the models M_0 ... M_{t-2} constructed by selecting the distinctive looking ones. A new model M_t is added to the set of canonical models if it does not match any existing canonical model well, using the bidirectional Hausdorff distance $H_{KL}(A, B)$.

Once the object is located, the model M_{t-1} is updated to build the new model M_t. The new model consists all edge pixels in I_t which are close (within a distance δ_U) to points in the previous model overlaid at the new object location, $M_{t-1} \oplus t^*$ as follow:

$$M_t = \{q \in I_t | \min_{p \in M_{t-1} \oplus t^*} ||p - q|| < \delta_U\}. \tag{2.15}$$

Finally, each border of the new model's bounding box is adjusted inward if there are relatively few non-zero pixels nearby or outward if there are a significant number of non-zero pixels nearby. Larger values of δ_U allow the tracker to follow non-rigid motion better and to pick up new parts of the object that come into view. However, as δ_U becomes larger it also becomes more likely that remaining background pixels will be incorporated into the model.

One final detail of the model update step is that we typically select at random a subset of the points given by Equation 2.15 for use as the new model M_t, instead of using every point selected by this equation. Given a model density factor σ_M, each point in M_t is selected with probability σ_M to be included in a sparse model M'_t, and the sparse model is used instead of the full model to locate the object in the next image frame and to generate a new model in the next time step. A sparser model reduces the effect of background pixels getting included in the model and also speeds the computation. This detail is not described in [10] but was discovered through reading the code. The logic behind this aspect of the algorithm was confirmed by talking to the author of a second generation version of the tracker [3].

Figure 2-9 demonstrates the Hausdorff tracker in operation. The edge detector output is shown in (a) and the edge pixels which remain after the background is removed are shown in (b). The previous model M_{t-1} is shown in (c). The updated model M_t is shown in (d). Finally, the model M_{t-1} is shown overlaid on image I_t demonstrating the match in (e).

The values of the parameters to this algorithm that we have used are those which were found to work well by the authors of [10]. Values for the major parameters of the algorithm are as follows: The match threshold δ_M is 10 pixels. The forward fraction f is 0.8. The forward fraction determines the number of model pixels which must match well, K, in the Hausdorff distance equation. Recall that $K = \lfloor fq \rfloor$, where q is the number of pixels in the model. The model update threshold δ_U is 20

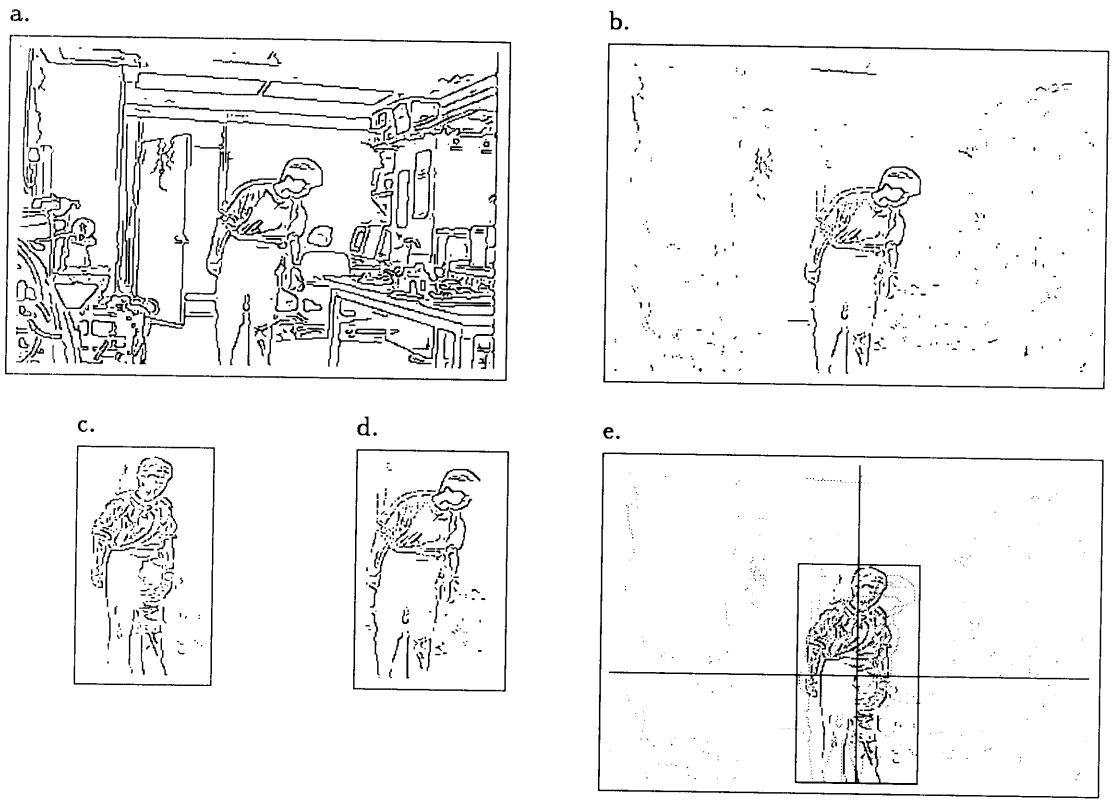


Figure 2-9: Example of the Hausdorff tracker in operation. (a) shows the edge detector output, (b) shows the image I_t after removal of stationary background, (c) shows the previous model M_{t-1}, (d) shows the new model M_t, and (e) shows the match of model M_{t-1} overlaid on image I_t.

pixels. Finally, the model density factor σ_M is 0.10 unless otherwise specified. We will consider the effect of varying this parameter in the experiments.

Chapter 3

Experimental Design

Having described the algorithms we will be investigating, we will now proceed to discuss the design of the experiments we will be performing in order to analyze the strengths and weaknesses of these algorithms.

We are interested for now at least in designing a person tracking algorithm to observe the room through a single camera using a wide-angle lens. The camera will have a fixed position as well as fixed focus and focal length. Someday, we would like to incorporate multiple cameras for person tracking and may even want them to be actively computer controlled but for now we consider only this restricted case.

Since we have chosen to have the camera fixed, it is easy to do the analysis off-line. Because the algorithms will always get the same views and can do nothing to affect the images they receive, they can be fed pre-filmed image sequences. This will greatly simplify testing because it enables repeatability. The exact sequence can be re-run as many times as needed. The algorithm's parameters may be varied or the algorithms may be further instrumented to gain clearer insight into what is happening. Also, running the algorithms off-line allows us to relax the real-time constraint. We can step frame by frame for close analysis and are not required to find platforms capable of running the algorithms in real time, as long as we can run them at reasonable speeds. A disadvantage, however, is that we will be limited by the amount of data we can reasonably store; we will not be able to test on many hours of video.

In our experiments we will be using bounding boxes for the locations of the objects being tracked rather than some more precise representation such as the set of image pixels corresponding to the object. There are several reasons for this choice. First, we are only interested in a rough segmentation of the scene which will allow the higher level processes to focus their attention on those areas where the people are known to be. At this low level, it is not necessary to track the arms and hands precisely, for example. This will be left for the gesture recognition system, which will probably

only want to devote such computational resources to the single person who currently 'has the floor' and is commanding the room. Bounding boxes provide a very easy to manipulate and compact representation. They also make it easy to compare two segmentations. Finally, bounding boxes make it easy to hand segment the images for scoring the tracker output against the correct answer.

Finally, we will be primarily looking at images sequences in which there is a single object to track. Some of the sequences do have small amounts of motion other than the primary object. None of the three algorithms are actually set up to simultaneously track multiple objects. They can be separately initialized to track each of a number of objects in the scene, however they have not been designed to deal with multiple objects in motion which can occlude each other. It is our primary goal to understand the features employed by each algorithm to understand how they might be modified and or combined to actually track multiple objects.

3.1 Image Sequences

We now describe the seven video image sequences we will use for our testing. They have all been grabbed at half resolution in both the x and y dimensions. Due to hardware limitations, it was not possible to grab images at sufficient rates at full resolution and there would not have been sufficient storage either. The downsampling in the x dimension was done by sampling every even pixel of each row and the sampling in y was done by selecting even fields only. Selecting only the even fields was necessary to eliminate motion blur caused by the odd and even fields being captured 1/60th of a second apart. No smoothing was done prior to downsampling.

The sequences were mostly captured using the NTSC color input of an SGI Indy computer and stored as 24 bit RGB color images. The one exception is the **takeh** sequence which was captured using a VideoPix board in a Sun SparcSTATION 2 and stored as 8 bit grey-scale images. We unfortunately did not have access to a suitable 24 bit color frame grabber. The SGI captured images are 320×240 pixels and were taken using the NTSC color output of a Pulnix TMC-50 RGB CCD camera. The VideoPix captured images are 360×240 pixels and were taken from a Sanyo CCD grey-scale camera. Frame rates given below for both frame grabbers are approximate. For example, 7.5 frames per second means that for the most part every fourth even field was captured. However, paging of the computers virtual memory system and other effects cause frames to be missed occasionally.

The image sequences we will use for our testing fall into two classes. Four contain rigid objects undergoing a controlled motion. These motions were made with a computer controlled linear positioner. The positioner was arranged horizontally on a table, and pulleys were used to route two thin threads from the moving platform to the stationary end of the positioner and up to the ceiling. The objects were hung such that they moved vertically as the positioner was controlled to move back

Sequence Name	Frames	Image Size (pixels)	Lens	Rate (frames/s)	Object Size (pixels)	Depth (inches)
can4	50	320x240	16mm	7.5	53x28	68
can5	50	320x240	16mm	7.5	53x28	68
cereal1	50	320x240	16mm	7.5	54x73	84
squares1	50	320x240	16mm	7.5	85x88	84
takeh	100	360x240	8.5mm	7.5	75x180	125
kazu	50	320x240	8.5mm	15.0	70x160	150
magda	50	320x240	8.5mm	15.0	60x130	150

Table 3.1: Image sequences

and forth. The cameras were turned sideways to give sequences in which the objects move from side to side, as the horizontal dimension has a wider field of view. The motion being made in these sequences is a constant acceleration of 18 cm/s^2 for 2 seconds (36 cm travel) followed by an identical deceleration to rest. The sequence of acceleration and deceleration then repeats. See Figure 3-1 for an example of a typical inter-frame-displacement profile for these sequences.

The remaining three sequences contain uncontrolled motion made by non-rigid objects. Each contains a single person walking around in a cluttered scene. Our only instructions were to stay within the field of view of the camera and not to get too close to the camera.

A short description of each sequence follows. Table 3.1 summarizes some information on the sequences, including the length in frames, the image size, the lens used, the frame rate, the approximate size of the imaged object, and the depth of the object in the scene. Figure 3-2 shows a few frames from each sequence.

The can4 sequence contains a "Dr. Pepper" soda can which moves from right to left across the center of the field of view, pauses momentarily, and then begins moving back to the right. The can also moves down somewhat while traveling to the left as the camera was not precisely aligned to the direction of motion. The can moves 233 pixels to the left and 20 pixels down in the course of one half cycle. The motion is roughly constant acceleration and deceleration as described previously. In this and all of the controlled motion sequences, the object is stationary for the first few frames of the sequence, then begins its motion. The can is dark red, with white lettering. The background contains three pieces of white poster board and part of the off-white wall. The can is relatively shiny and in the first 10 frames there is a strong specularity.

The can5 sequence is similar to to the can4 sequence except that the background behind most of the can's motion has been replaced by a print of "Starry Night" by Van Gogh. This provides a highly textured and brightly colored background to compare the effect of background clutter. The can moves 232 pixels to the left and 20 pixels down in the course of one half cycle of the motion.

The cereal1 sequence is another controlled motion using the same constant acceleration profile. The object in motion is a cereal box with a brightly colored image and text. The surface is not very specular. Again there is a small amount of vertical motion. The box moves 194 pixels to the left

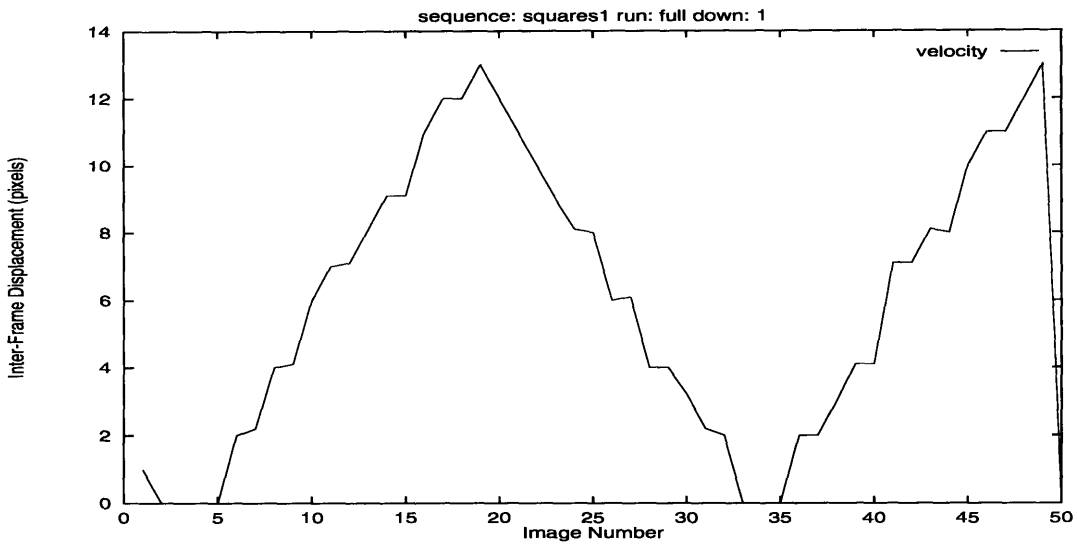


Figure 3-1: Sample velocity profile for the `squares1` sequence, showing the constant acceleration and deceleration controlled motion (linear velocity profile). The horizontal axis shows the frame number and the vertical shows the magnitude of the inter-frame displacement of the object, in pixels.

and 16 pixels down in the course of one half cycle of the motion.

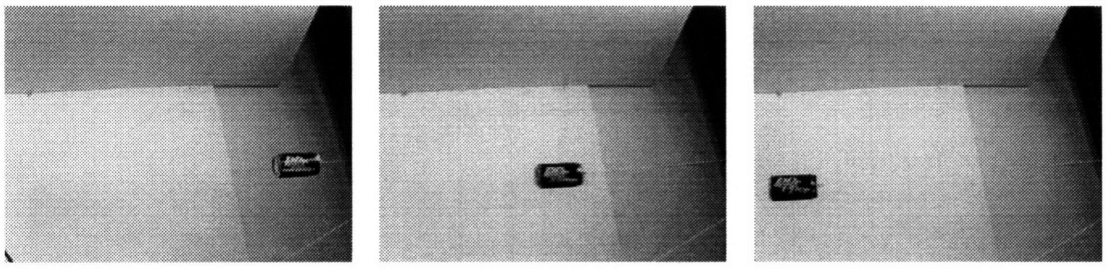
The `squares1` sequence is the final sequence using the controlled constant acceleration motion. A piece of cardboard with four brightly colored squares of construction paper moves from right to left and then begins moving back to the right. The upper left square is red, the lower left is blue, the upper right is green, and the lower right is yellow. This sequence was designed to test the effect of an untextured object. As in all the controlled motion sequences, there is a small amount of vertical motion. The squares move 192 pixels to the left and 17 pixels down in the course of one half cycle of the motion. Figure 3-1 shows a graph of the magnitudes of the inter-frame displacements for each frame of this sequence. Notice that it is stationary for a few initial frames before begining its motion. It is also stationary for a few frames when it reaches the left edge of its motion before starting to move back to the right. This profile is representative of all the controlled motion sequences.

In the `takeh` sequence, a man walks around our cluttered vision lab. Someone is working on a computer in the background. This sequence is the only grey-scale (intensity only) sequence. The man is wearing a light solid colored shirt and dark pants.

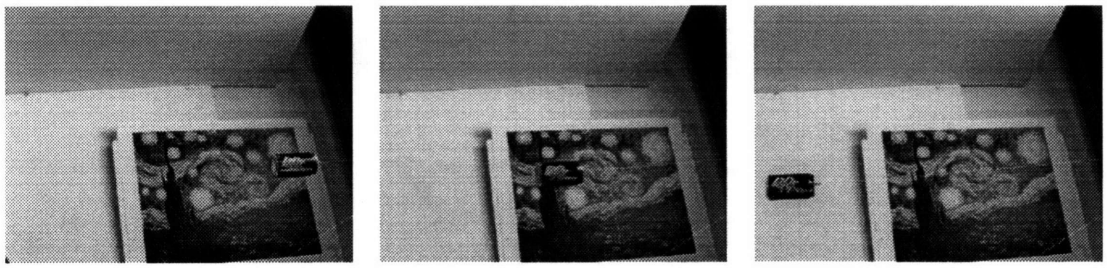
The `kazu` sequence has another man walking around in the end of the Intelligent Room set up as a computer lab. The computers, furniture, and mass of cables on the wall make the background very complex. The man is wearing a dark blue T-shirt and dark blue jeans.

The `magda` sequence has exactly the same background as the `kazu` sequence. They were taken minutes apart with the same camera position. In this sequence a woman walks around in the room. She has long hair, is significantly shorter, and is wearing a white and blue horizontally striped shirt and blue jeans.

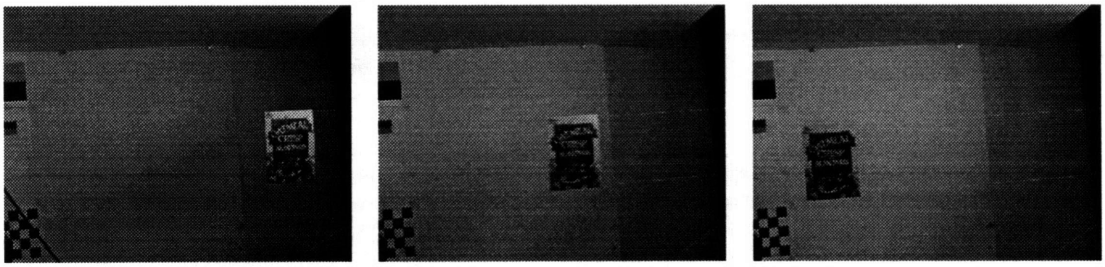
can4 sequence, frames 0, 17, and 34



can5 sequence, frames 0, 17, and 34



cereal1 sequence, frames 0, 17, and 34



squares1 sequence, frames 0, 17, and 34

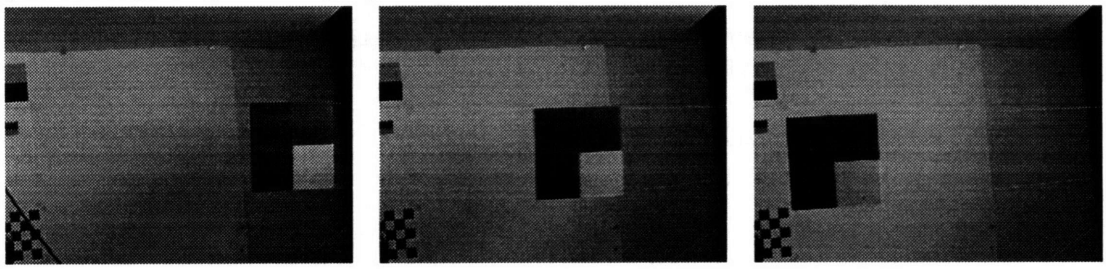
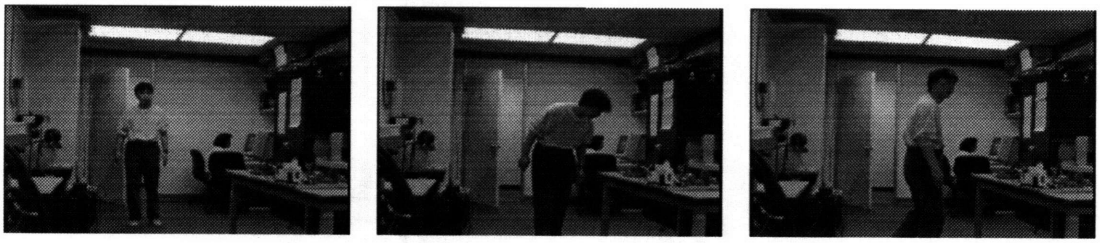
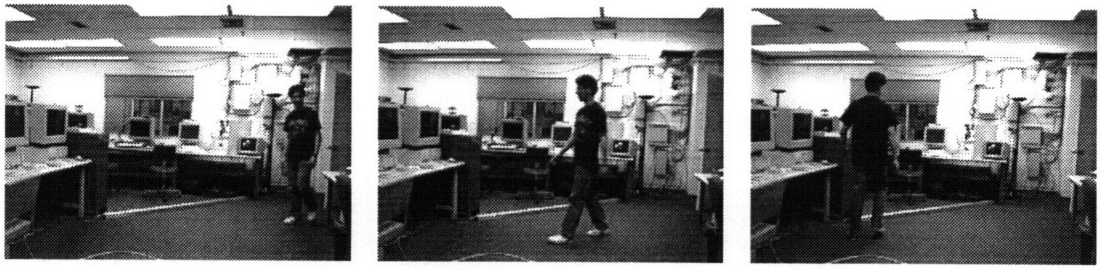


Figure 3-2: Sample images from the can4, can5, cereal1, and squares1 sequences.

takeh sequence, frames 0, 19, and 92



kazu sequence, frames 0, 25, and 48



magda sequence, frames 0, 25, and 48

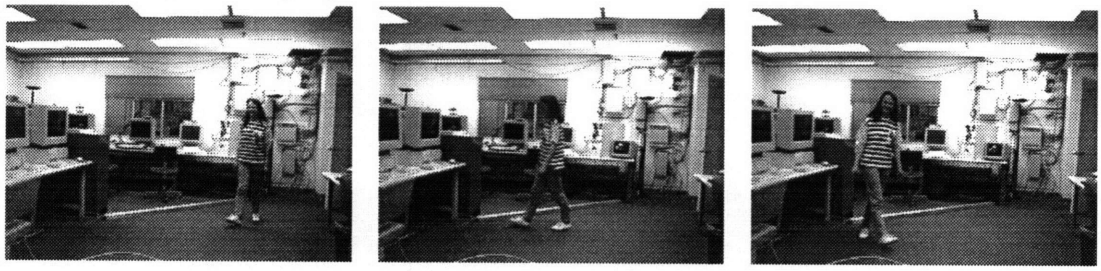


Figure 3-2: (cont.) Sample images from the takeh, kazu, and magda sequences.

3.2 Experiments

The experiments we will be performing consist of a number of situations we expect to encounter, as well as varying algorithm parameters. We will be investigating the effects of background clutter, object texture, speed of motion, including lack of motion, lighting changes, and object size. We will vary the image resolution, and test the effect of downsampling with and without pre-smoothing with an anti-aliasing filter. We will be testing the effect of rigid versus non-rigid objects as we are interested in tracking humans. Further, we will be testing several prediction techniques for estimating the new feature location for the correlation based feature tracker. We will vary the model density factor σ_M in the Hausdorff algorithm to determine the effect of this parameter. Finally, we will test the shape discrimination ability of the Hausdorff tracker to determine if it is capable of solving the person constancy problem for tracking multiple people.

An experimental run will consist of an image sequence, a tracking program to use, and any options. These options include program parameter values, an ordered subsequence of the image frames to use, spatial and temporal downsampling factors and method, and artificially simulated lighting changes. Hence, each image sequence can give rise to many experiments. Given that we have 3 algorithms, 8 sequences (an eighth will consist of images from both the `kazu` and `magda` sequences), 5 spatial downsamplings, and at least 20 option settings, there are many thousand experimental runs under consideration. We have not done every one, but the 667 that we have done constitute more than 25 percent. We will see below how we are able to easily and efficiently analyze all of this data.

3.3 Ground Truth and Scoring

Each frame of each image sequence has been hand segmented to determine a ground truth bounding box for the object in motion. A simple graphical ground truth editor has been written to make this process as quick and easy as possible. The segmentation has been done with some care, however we did not attempt to obtain single pixel precision as it will not make a significant difference in the scoring described below. The segmentations are for the most part accurate to one or two pixels for each vertex of the bounding boxes. Figure 3-3 shows the segmentation of a few frames from the `takeh` sequence.

Given the ground truth segmentation for a sequence and the output of one of the tracking algorithms on an experimental run, we can score the output for each frame against the ground truth segmentation for that frame. Since our representation is simply a bounding box, it is quite easy to compare the two segmentations.

We need a score function $S(B_1, B_2)$ which compares the similarity of two bounding box segmentations, B_1 and B_2. It will be convenient if the score is normalized between zero and one, with a value of one if the boxes are identical and zero if they are not at all similar. The function should also

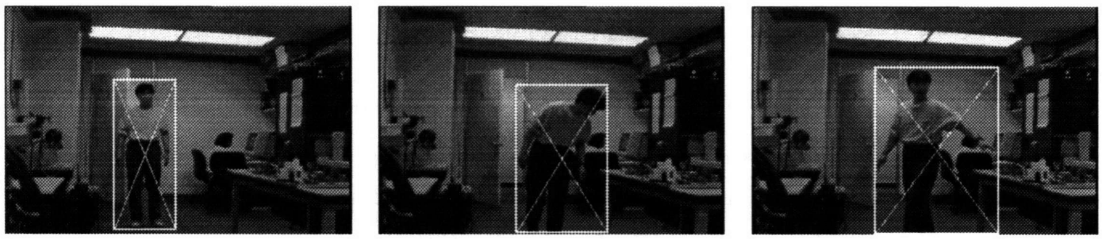


Figure 3-3: Ground truth segmentation for frames 0, 17, and 75 of the **takeh** sequence. Crosses in the bounding boxes are drawn only for emphasis.

be symmetric. One such function is given by the overlap between the two boxes divided by their average area. If we represent a bounding box by a 4-tuple containing its left and right x coordinates and top and bottom y coordinates

$$B = \langle x^l, x^r, y^t, y^b \rangle$$

then we can write this scoring function as

$$S_O(\langle x_1^l, x_1^r, y_1^t, y_1^b \rangle, \langle x_2^l, x_2^r, y_2^t, y_2^b \rangle) = \frac{2\,V(x_1^l, x_1^r, x_2^l, x_2^r) \cdot V(y_1^t, y_1^b, y_2^t, y_2^b)}{A(\langle x_1^l, x_1^r, y_1^t, y_1^b \rangle) \cdot A(\langle x_2^l, x_2^r, y_2^t, y_2^b \rangle)} \tag{3.1}$$

where

$$V(a_1, b_1, a_2, b_2) = \max(\min(b_1, b_2) - \max(a_1, a_2) + 1, 0)$$

is the overlap in a single dimension and

$$A(\langle x^l, x^r, y^t, y^b \rangle) = (x^r - x^l + 1) \cdot (y^b - y^t + 1)$$

is the area of a bounding box. We have found this simple function to be sufficient for our purposes. It gives values close to one if the two boxes are roughly the same size and overlap significantly and falls off toward zero as these conditions are not met.

One complication comes from the fact that the correlation based feature tracker does not output a bounding box for the whole object being tracked. This is because it is only meant to track a feature point, and has no notion of the whole object. Instead, it outputs the location of the feature point which it is tracking. The bounding box corresponding to the feature patch centered at this point could be used for the score function above, however, the scores are likely to be low since the feature point is often just a small patch of the object being tracked. Hence, we need to use a different scoring function for this algorithm to compare the point location output by the feature tracker to the object bounding box. We have chosen to use a normalized distance from the center of the bounding box which is one at the center, falls to zero at the corners, and is zero outside the box. Representing the point as

$$P = \langle x, y \rangle$$

and the box as

$$B' = \langle x^c, y^c, r^x, r^y \rangle$$

where x^c and y^c are the center coordinates and r^x and r^y are the half width and half height, respectively, we can define this scoring function as

$$S_{NI}(P_1, B'_2) = (1 - S_N(P_1, B'_2)/\sqrt{2}) \cdot S_I(P_1, B'_2) \tag{3.2}$$

where

$$S_N(\langle x_1, y_1 \rangle, \langle x_2^c, x_2^c, r_2^x, r_2^y \rangle) = \sqrt{\frac{(x_1 - x_2^c)^2}{(r_2^x)^2} + \frac{(y_1 - y_2^c)^2}{(r_2^y)^2}} \tag{3.3}$$

is the distance of the point from the center of the box, normalized by the box dimensions, and

$$S_I(P, B') = \begin{cases} 1 & \text{if } P \text{ is inside } B', \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

tells whether the point is inside the box or not. Note that this second bounding box representation is derived from the original by

$$\begin{aligned} x^c &= (x^l + x^r)/2 \\ y^c &= (y^t + y^b)/2 \\ r^x &= (x^r - x^l + 1)/2 \\ r^y &= (y^b - y^t + 1)/2. \end{aligned}$$

We have found this function to be sufficient for scoring the runs using the feature point tracker. Unless otherwise noted, all scoring for the motion based and Hausdorff tracking algorithms will be scored with the overlap score function, $S_O(\cdot, \cdot)$ given by Equation 3.1 and all scoring for the feature point tracker will be done using the normalized-inside score function $S_{NI}(\cdot, \cdot)$ given in Equation 3.2. This makes it difficult to directly compare tracking quality of the correlation tracker with the other two, but this has not been a problem in our experiments.

An overall score can also be computed for a run by simply taking the average of the scores for each of the image frames. Below we will discuss how we will make use of these frame by frame and overall run scores.

3.4 Analysis

To analyze an experimental run, we begin by scoring against the ground truth segmentation and then plot the frame by frame scores as a function of image number. Viewing these plots makes it

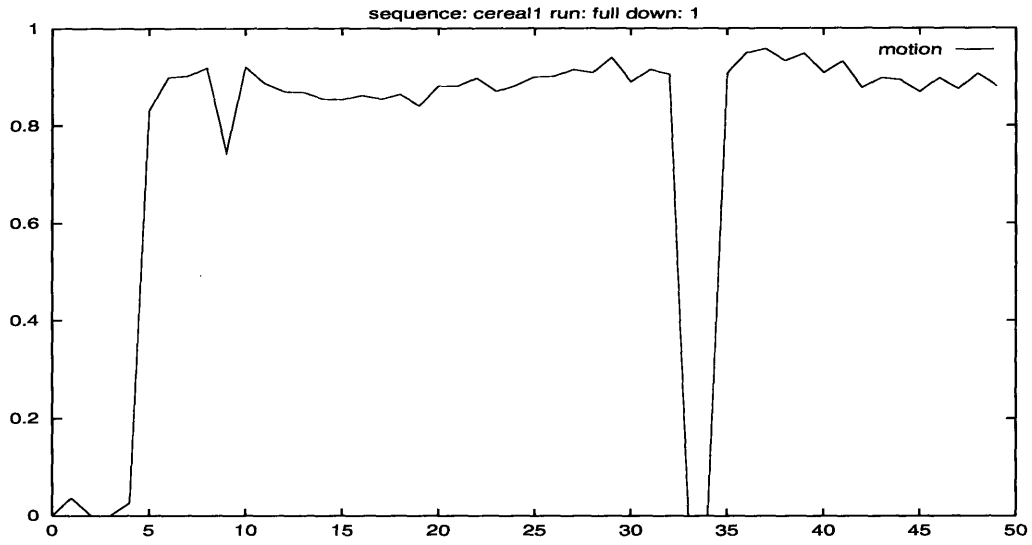


Figure 3-4: Example of the frame by frame scores plotted for a run of the motion based tracker on the `cereal1` sequence at full resolution.

easy to determine if the tracking was successful and if not, in which frame the problem started. This allows us to very quickly focus our attention on the experimental runs and image frames within those runs where problems were encountered. We can then step the algorithm a frame at a time at these places to better understand why the algorithm is failing. The three trackers are instrumented to show the intermediate stages of the computation visually so it is usually easy to determine the problem.

Figure 3-4 shows a plot of the scores for a run of the motion based algorithm on the `cereal1` sequence. It is clear that the algorithm is tracking well, except for the first few frames, and for frames 33 and 34. We will see in the next chapter that this is due to a lack of any image motion at these times. During the first few frames, the object has not yet begun moving, and in frames 33 and 34 it comes to rest before reversing direction.

Plotting the scores also allows us to compare performance between experimental runs due to the different algorithms, image sequences, and parameters settings. For example, Figure 3-5 demonstrates that on the `squares1` sequence, the correlation based feature tracker does progressively worse as the resolution is decreased from full resolution to half and then quarter resolution. However, the tracking is still quite good even at this lowest resolution, never dropping below a score of 0.6.

Occasionally it is also useful to plot overall scores for experimental runs when we wish to consider varying two independent variables at the same time. See Figure 4-26 in Section 4.5 for an example.

We will be using plots like these extensively in the next chapter to analyze the results of many experimental runs.

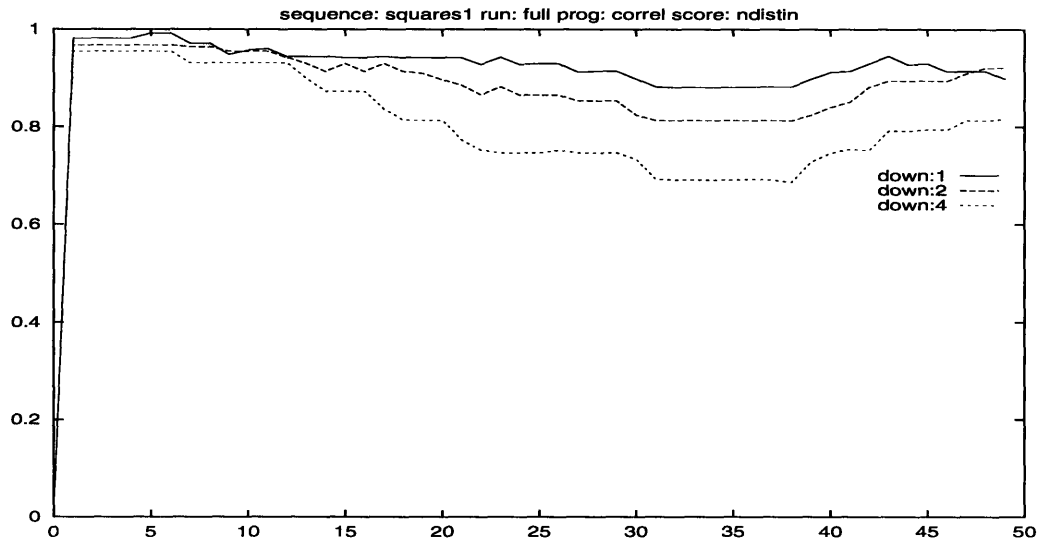


Figure 3-5: Example plot comparing the frame by frame scores of runs of the correlation based feature tracker on the `squares1` sequence at full, half, and quarter resolutions.

Chapter 4

Experimental Results

In this chapter we will describe all of the experiments we have done on our tracking algorithms and the results of those experiments. The experiments will be described for the most part chronologically so that the reader may understand the reasoning behind the experiments chosen. We begin by looking at the initial runs of the algorithms on our sequences.

4.1 Initial Experimental Runs

The initial set of experiments consisted of running each of the three algorithms described in Chapter 2 on each sequence in full. No sampling of the sequences was done in time, and all frames of each sequence, starting with the first, were used. Only the first five sequences described in Section 3.1 were used, namely the `can4`, `can5`, `cereal1`, `squares1`, and `takeh` sequences. The final two sequences had not been captured at this time (see Section 4.7).

Each sequence was run at three spatial resolutions: full, half, and quarter resolution. Full resolution is the resolution at which the sequence was captured, as listed in Table 3.1. Note that this is really half the resolution output by the cameras as detailed in Section 3.1. All references to image resolution in this chapter are with respect to the size at which they were captured. The half and quarter resolution images were obtained by sampling the full resolution images by factors of two and four in each spatial dimension, respectively. No pre-smoothing anti-aliasing filter was applied. This alternative will be considered below in Section 4.4. Hence, given a full resolution image $I[x, y]$ the half and quarter resolution images, $I_{\text{down2}}[x, y]$ and $I_{\text{down4}}[x, y]$ are simply given by

$$I_{\text{down2}}[x, y] = I[2x, 2y] \quad \text{and} \quad I_{\text{down4}}[x, y] = I[4x, 4y]. \tag{4.1}$$

Hence, there were 45 runs in this first set of experiments. The results for each algorithm are described below.

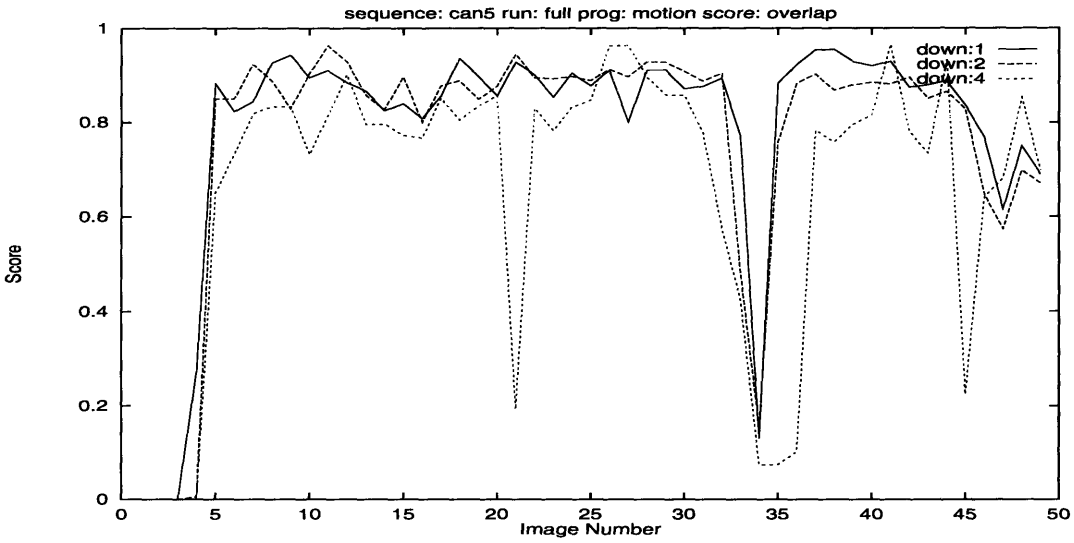


Figure 4-1: Scores plotted for the motion based algorithm run on the can5 sequence at three resolutions.

4.1.1 Motion Based Algorithm

In these experiments, the motion based tracking algorithm performed quite well overall. Figure 4-1 shows a plot of the scores for the can5 sequence at each resolution. Except for a few short periods, the tracking is very good, with scores above 0.8.

Since the algorithm is based on finding connected regions of image motion, we expect that a lack of motion should cause problems for the algorithm. There is no motion in frames 0 through 3 because the can has not yet begun to move and we can see that the tracking has failed here. In frames 33 and 34 as the can slows to a stop to change direction there is again negligible motion. We see in the graph that the tracking has failed almost completely in these frames. As the image velocity approaches zero, fewer and fewer motion pixels are detected and the box bounding them shrinks to nothing. Even if the object does not completely stop but is moving at sub-pixel displacements the tracking suffers drastically. Since the optical flow detects discrete motion displacements it very often detects no motion at a given pixel in this case, giving a very sparse set of motion pixels at the object's location which again often shrinks down to almost nothing. Figure 4-2 shows a closeup of this condition as the can is coming to a stop in frames 31 to 33 of the half resolution run. It is evident that very few motion pixels were detected even though the can is still moving. The sparseness of the detected motion pixels has further caused them to be labeled as two different connected components in the final frame. The algorithm then chooses the lower fragment as the tracked object.

Due to the optical flow algorithm's limited search radius (R_S) we also expect to have problems if the motion is too fast and pixels move more than this distance between frames. In this case the optical flow finds no match for a patch centered at a pixel in the previous image within the search region in the current image. In frames 21 and 45, the score for the quarter resolution run drops

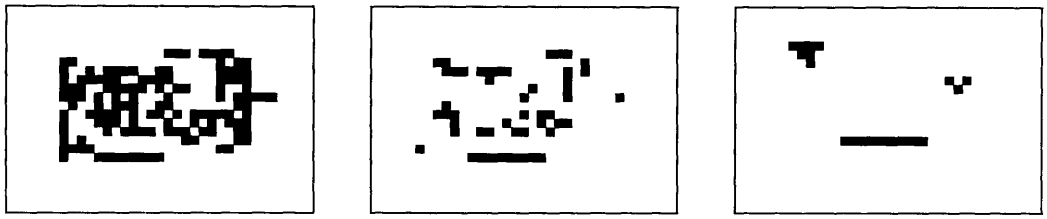


Figure 4-2: Closeup of the motion pixels detected as the can comes to a stop in frames 31, 32, and 33, shown left to right.

significantly due to exactly this problem. The image velocity is 4 pixels to the left per frame and the search radius is 3 pixels at that resolution so it is understandable that no matches should be found for many pixels. Motion is being detected for some pixels however, but they are very sparse and have broken into several components. That motion is detected at all may seem surprising. It is due to optical flow matches being found within the search radius at the wrong locations, but which are good enough to be considered valid.

Looking again at Figure 4-1 we notice that at quarter resolution the algorithm is affected the worst by a lack of motion as well as by motion which is too fast. In fact, the higher resolutions were not affected at all by the can moving too fast, even though the search radius is scaled linearly with size and so should have had the same problem. At the higher resolutions, there are many more pixels within the outline of the can and it is much more likely that enough will find incorrect matches to prevent fragmentation into multiple components. It is also the case at higher resolutions that the object has to be moving much more slowly before the motion is in the sub-pixel per frame regime. For example, if the can moves 2 pixels per frame at full resolution then at half resolution it is moving 1 pixel per frame and at quarter resolution only 0.5 pixel per frame. These effects cause the tracking to be better at higher resolutions.

The tracking was quite bad, however, on the `squares1` sequence as shown in Figure 4-3. In addition to the rather minor problems seen above, the lack of texture on the colored squares has caused serious problems in the difference image and optical flow steps. A closeup of the detected motion pixels for image 20 of the half resolution run is shown in Figure 4-4. Motion has only been detected in three vertical bands due to there being no texture on the colored squares. Because the squares are uniform color and brightness, as they move left a few pixels, a change is detected in the difference image only at the left and right edges and the vertical center boundary between the colored squares. Recall that clockwise from the upper left the squares are colored red, green, yellow, and blue. At the left edge, a difference is detected between both the red square and the background and between the blue square and the background. At the right edge, change is detected between the green square and the background, but no change is detected between the yellow square and the background because they have similar intensities. In the center, change is detected between the blue and yellow squares but not the red and green squares which have similar intensities. No change

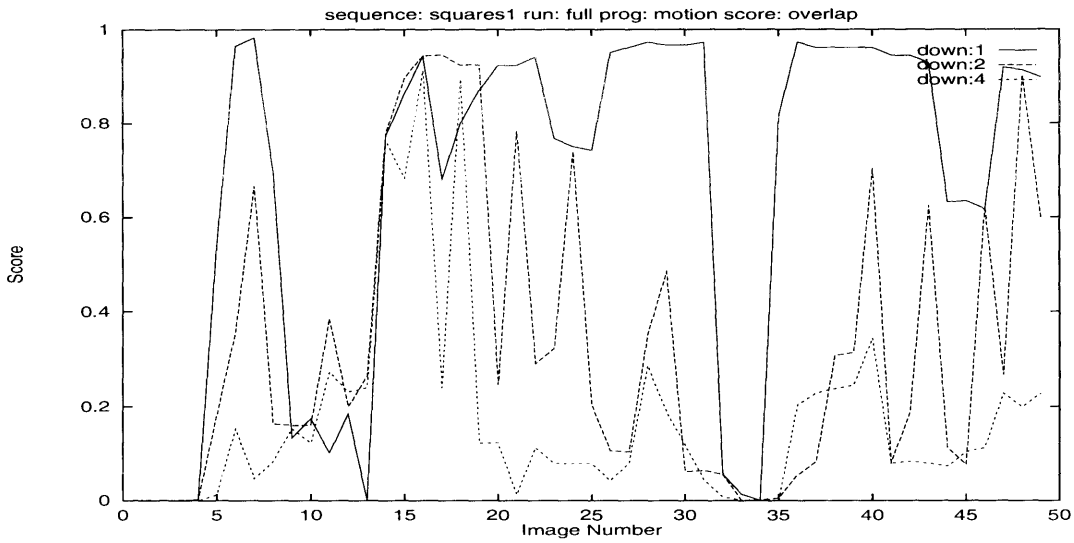


Figure 4-3: Scores plotted for the motion based algorithm run on the `squares1` sequence at three resolutions.

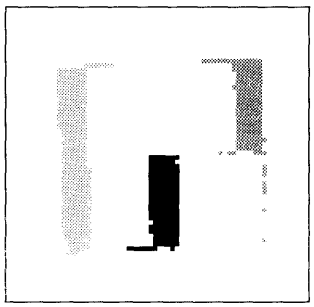


Figure 4-4: Fragmentation of the detected motion pixels due to an untextured object in the `squares1` sequence. This is a closeup of image 20 run at half resolution.

is detected in the interior of the squares where a pixel has been replaced by a pixel of the same color and intensity that had been slightly to the right. Since we only compute optical flow at the difference locations, no motion is detected over much of the squares, leading the motion regions to be separated into three separate connected components and giving poor tracking of the whole object. Hence, we can see that lack of texture on the objects we are tracking is a serious problem for the motion based tracker. The problems are worse as the resolution decreases because there are fewer motion pixels which are more likely to fragment into many components.

We encounter the same problem with lack of texture again in the `takeh` sequence. Figure 4-5 shows the tracking results for the three resolutions. The man's clothing is sufficiently untextured that motion is only getting detected at the boundary between the man and the background, and not in the interior of his body. The motion pixels detected in frame 27 in the half resolution run are shown magnified in Figure 4-6 and show this condition. Again, the motion pixels are not getting grouped as a single component in many cases due to the lack of dense motion information. From

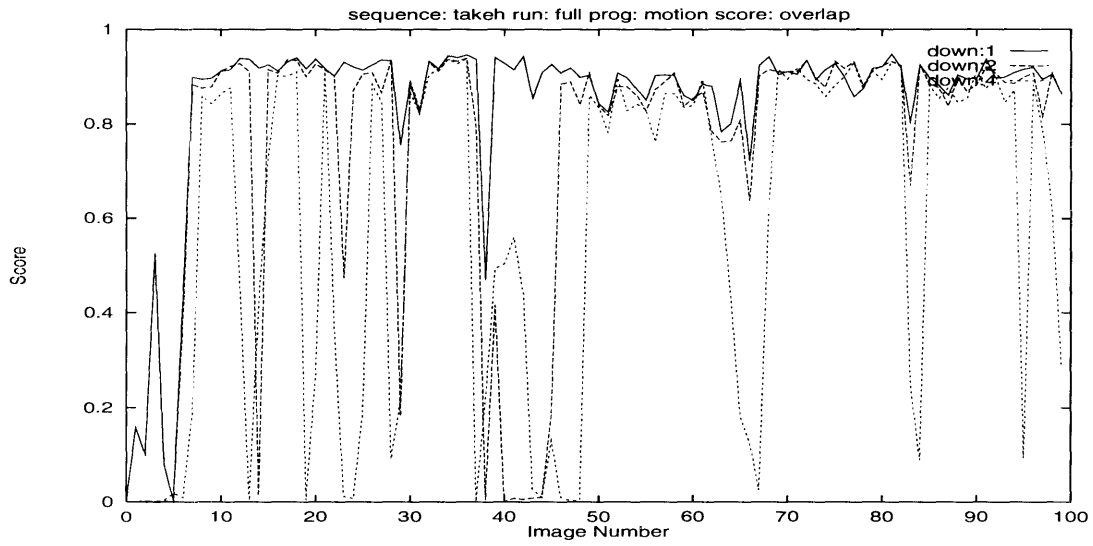


Figure 4-5: Scores plotted for the motion based algorithm run on the **takeh** sequence at three resolutions.

Figure 4-5 we see that this problem was not significant at full resolution but at half and quarter resolution it adversely affected the tracking considerably.

These experiments show that the motion based tracker works well for a textured object moving at reasonable speeds. However, it has problems with very slow and very fast motion as well as untextured objects. These problems are exacerbated by decreasing resolution. They typically result in the object breaking into several motion regions. In the case of the object completely stopping, the object disappears from the motion segmentation and ceases to be tracked until it resumes motion. However, if a second object which had stopped were to start moving, we would resume tracking that one instead. There is no notion that the object being tracked is the same one we were previously tracking, except by it being the nearest moving thing to the object's previous location.

4.1.2 Hausdorff Algorithm

In all of the initial runs on the sequences containing controlled motion, the Hausdorff algorithm performed poorly. The scores are plotted for the run on the `cereal1` sequence in Figure 4-7. This shows one of the better runs for this algorithm. At full resolution, the score is zero up until frame 12, then rises into the 0.60–0.80 range until frame 34, at which time it drops back to zero and then jumps around wildly. The lack of any motion in the first 5 frames of these sequences is the primary cause of the tracking problem. In frame zero, 1016 pixels are found in the matching process and the sparse model M_0 output by the algorithm in that frame (at 0.10 model density) had 91 pixels. However, only 110 pixels are matched in the next frame because the object was stationary and the background removal got rid of most of the edge pixels belonging to it. The new model M_1 has only 5 pixels (recall that the sparse model is made by independently selecting each pixel with probability

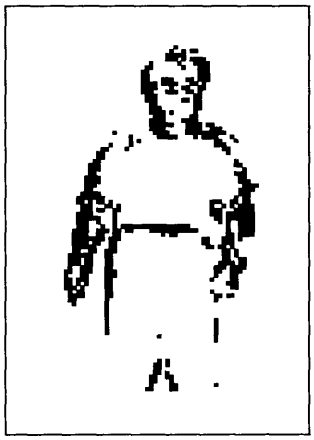


Figure 4-6: Due to the man's untextured clothing, motion is only detected around the outside of the man the takeh sequence and not in the interior. This closeup shows the motion pixels detected in image 27, run at half resolution. The motion pixels have fragmented and have been grouped into two connected regions.

0.10). This is few enough pixels that in frame 2 it matches a few background pixels which did not get cleaned by the background removal. The model M_2 then has only seven pixels and matches some other remaining background and this process continues through frame 11 with the algorithm jumping around between bits of remaining background. Eventually, in frame 12, it happens to match best on the cereal box and begins tracking the cereal box. The problems encountered in frame 34 are due to the box stopping its motion briefly. The causes are similar as we will see in Section 4.2 below.

We did not have this problem running on the takeh sequence because there is motion throughout the sequence. Figure 4-8 shows the scores for this run. At full resolution, the man has been tracked very well throughout the sequences. At half resolution, however, the motion in the first couple frames is slow enough (being half that in the full resolution sequence) that we have the same problem encountered in the cereal1 sequence. The number of model pixels falls very low and the tracker begins erratically tracking remaining background pixels. After jumping back into the edge pixels corresponding to the man, it tracks him well throughout the rest of the sequences. At quarter resolution, the tracker does not function at all on this sequence.

Thus, we have seen that a lack of motion is a major problem for the Hausdorff algorithm, particularly when just beginning to track an object. We did see, however that with sufficient motion it is able to track the highly non-rigid motions of a person very well, at least at full resolution.

4.1.3 Correlation Feature Tracking Algorithm

The correlation feature tracking algorithm performed very well on the majority of the initial experiments. Figure 4-9 shows the tracking results for the cereal1 sequence at each resolution. Another example of the tracking from these experiments was given in Figure 3-5. In both examples, the

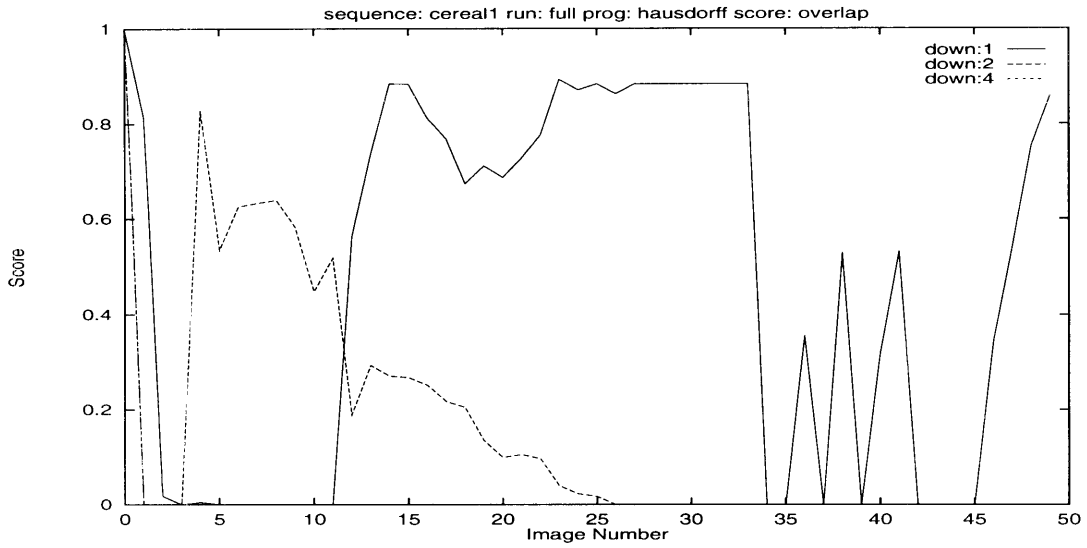


Figure 4-7: Scores plotted for the Hausdorff algorithm run on the `cereal1` sequence at three resolutions.

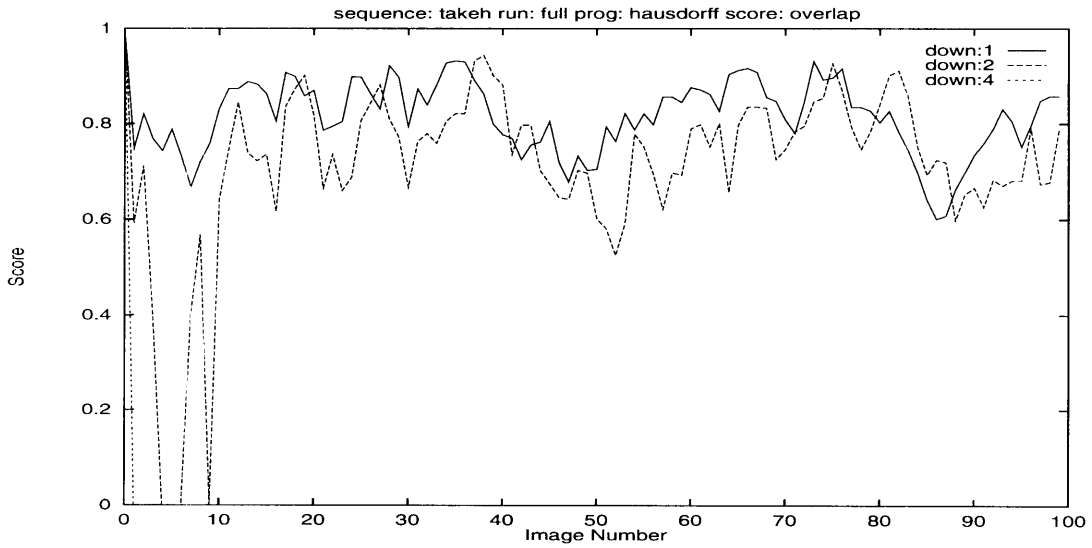


Figure 4-8: Scores plotted for the Hausdorff algorithm run on the `takeh` sequence at three resolutions.

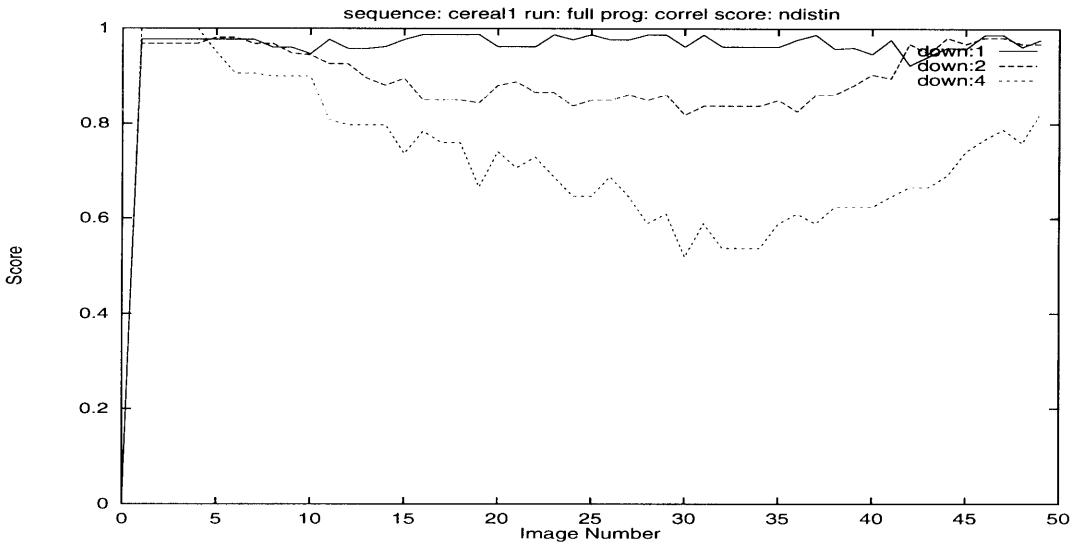


Figure 4-9: Scores plotted for the correlation based feature tracking algorithm run on the `cereal1` sequence at three resolutions.

tracking is very good. For the `cereal1` sequence at full resolution, the scores are above 0.95 for nearly the whole sequence. Even at quarter resolution, the scores are above 0.6 for most of the sequence. This is still quite a good score for the normalized-inside score function (Equation 3.2) used with the feature tracking algorithm. Performance on the `squares1` sequence is similar, though the scores were not as high for full resolution and not as low for quarter resolution. In both cases, however, there is a clear degradation in performance as the resolution is lowered.

A striking feature of the tracking on these sequences is that the algorithm is not affected by the initial lack of motion or the brief pause as the object stops in frame 35 and then reverses direction. The scores remain steady across these periods. Only the prediction of the new feature location is based on difference images in this algorithm. The actual feature patch matching works on the camera images directly without attempting to remove constant background. This is in contrast to the motion based tracker and the Hausdorff algorithm.

The plotted scores for these runs make it obvious that the algorithm is slowly drifting off of the object up until frame 35 and then begins to drift back toward being dead-on. The scores decrease steadily until frame 35 and then begin to increase as the object changes direction and moves back to the right. Close inspection reveals that as the objects are moving back and forth, they are also moving vertically very slightly. As the object moves to the left, it is also moving down between one half and one pixels per frame. This translates to between one quarter and one half pixel motion at half resolution and between one eighth and one quarter pixel motion at quarter resolution. Since the correlation only considers discrete, whole pixel motions, it must choose a vertical displacement of either zero or one pixel for each frame, whichever is closer to the true displacement. Hence, during all of the frames of both sequences, no vertical motion is detected in the half and quarter resolution

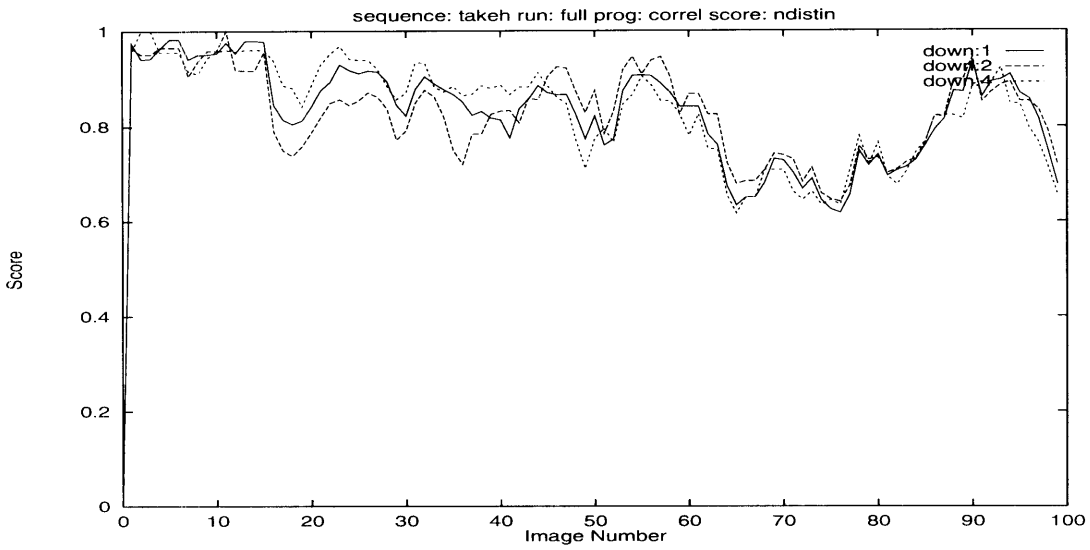


Figure 4-10: Scores plotted for the correlation based feature tracker run on the takeh sequence at three resolutions.

runs and only some of the vertical motion is detected at full resolution. The feature patch drifts off of the object vertically as it moves to the left, but then drifts back on after it changes direction, and the score increases. This problem is even worse in the can4 and can5 sequences. Because the can is so much smaller, the tracker drifts completely off of the can at quarter resolution and is not able to track it through the entire sequence. Note that once this algorithm drifts off the object there is no way for it to later re-acquire it. It simply locks onto a stationary patch of the background. The problem of sub-pixel motion is actually more general. It exists for any non-integer pixel motions, and makes it highly likely that as a feature is tracked over a long time the detected location will drift from the true location.

Looking at the tracking on the takeh sequence shown in Figure 4-10 it is obvious that the algorithm has performed very well on this sequence too. In fact the performance is nearly identical at all three resolutions. One can see a definite gradual downward trend in the scores through the hundred frames of the sequence, however. This points to the problem of the feature patch slowly drifting off of the feature being tracked, which we will discuss in more detail in Section 4.7. It is impressive that with the full range of non-rigid motion exhibited in this sequence that the algorithm was able to do so well at all resolutions.

One final insight we gained from these experiments relates to the motion difference prediction algorithm used for guessing the new feature location. Looking at the predictions made in the squares1 runs there is a significant downward bias to every prediction despite the fact that the object is moving almost entirely to the right, with at most a single pixel downward motion between frames. This bias ranges from 5 to 15 pixels throughout the sequence at full resolution. In each frame, the best match offset found always had the correct value to counteract this bias so it did not

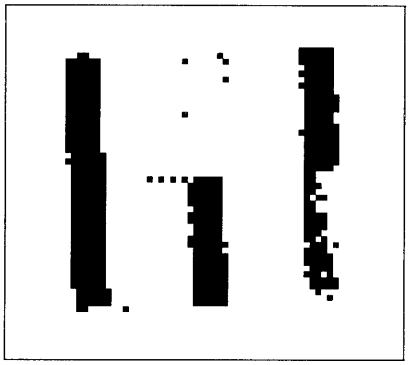


Figure 4-11: Closeup showing the motion difference pixels selected by the feature point tracker's prediction algorithm in frame 20 of the `squares1` sequence run at half resolution.

affect the tracking, however. Looking at the closeup of the motion difference from frame 20 of the sequence in Figure 4-11 we see that the problem is due to the untextured patches comprising the `squares1` object and the lack of a significant intensity difference between the red and green squares on top. As we saw with the motion based tracker, a difference is only detected at the boundaries of the colored squares, and no difference is detected between the red and green squares. Because a difference is detected only between the lower squares as the object moves, the prediction has a downward bias. We will analyze the shortcomings of the motion difference prediction algorithm in detail below in Section 4.9.

In summary, the initial experiments on the feature tracking algorithm have shown that it tracks very well on all of the sequences at all of the resolutions considered. It performed extremely well on the `takeh` sequence despite a large amount of non-rigid and non-translation only motion. The performance does typically degrade as the resolution is decreased. We observed problems due to sub-pixel motions causing the tracker to drift from the feature point which increase in severity as resolution decreases. Finally, we noted a problem with the prediction computation for non-textured objects.

4.2 Starting with Motion

As our second experiment we wanted to more meaningfully test the Hausdorff tracker on the controlled motion sequences. Recall that the lack of any motion during the first four to five frames of these sequences resulted in either very poor tracking or complete failure of the algorithm to track the objects. We decided to re-run each of the initial runs, but starting with frame 8 of each sequence (we will refer to these experiments as the '`start8`' runs). For completeness, the new runs were done on all three algorithms and at all three resolutions. The performance of the correlation based feature point tracker was similar to that described above and will not be discussed further.

4.2.1 Motion Based Algorithm

As expected, running the motion based algorithm starting with the eighth frame completely fixed the problem seen in the initial experiments due to a lack of motion when starting. The tracking throughout the rest of the frames was identical to that seen in the previous section.

4.2.2 Hausdorff Algorithm

Starting with the eighth image frame greatly improved the tracking for the Hausdorff algorithm. For the motion based algorithm, a lack of motion in the first few frames had no lasting effect once the object started moving because the algorithm is attracted to whatever is moving. The Hausdorff algorithm, on the other hand, was affected throughout the rest of the sequence by this condition because the object models were corrupted by shrinking to an unacceptably small number of pixels such that they matched leftover background edge fragments. This has such a lasting effect because the distinctive models are kept in the set of canonical models, used when no good match is found.

Figure 4-12 shows the tracking results for the `cereal1` sequence at each of our three resolutions. At full resolution the tracking is very good through the entire sequence, except at frame 26 where the score is zero. This is the frame in which the object had stopped moving and was about to reverse direction. Due to the stationary background removal, the box was not found. However, in the next frame, the model from frame 25, M'_{25}, successfully matched the cereal box and tracking resumed.

Looking closely, however, we noticed that the (sparse) model, M'_{25}, which matched and allowed the tracking to resume contained only seven edge pixels. As we saw in the initial runs above, this few pixels is hardly sufficient to guarantee that the algorithm will not readily match remaining background fragments. Hence, it was mostly by luck that the tracking was able to resume after frame 26. Because the box came to a stop slowly, the sequence of models gradually became sparser as more of the pixels from the box were considered stationary by the background removal. As this happened, the match scores given by Equation 2.14 in locating the box in successive frames stayed low enough to be considered a good match. This problem due to the object stopping is a serious shortcoming for this algorithm. Similar results were seen on the `squares1` sequence. Here, however, after appearing to recover from the lack of motion, the tracking quickly degraded. In fact, the algorithm was not actually able to recover from the model shrinking to only eight pixels in this case. In Section 4.5 we will investigate the effect of making the models less sparse by increasing the model density factor, σ_M. It would also be interesting to investigate an object stopping abruptly for comparison.

Comparing the tracking at the other resolutions, we see that at half resolution the algorithm does a poor job on the `cereal1` sequence, just barely tracking the box through the first half of the images. At quarter resolution the tracker is completely unable to track the box. This gives us some

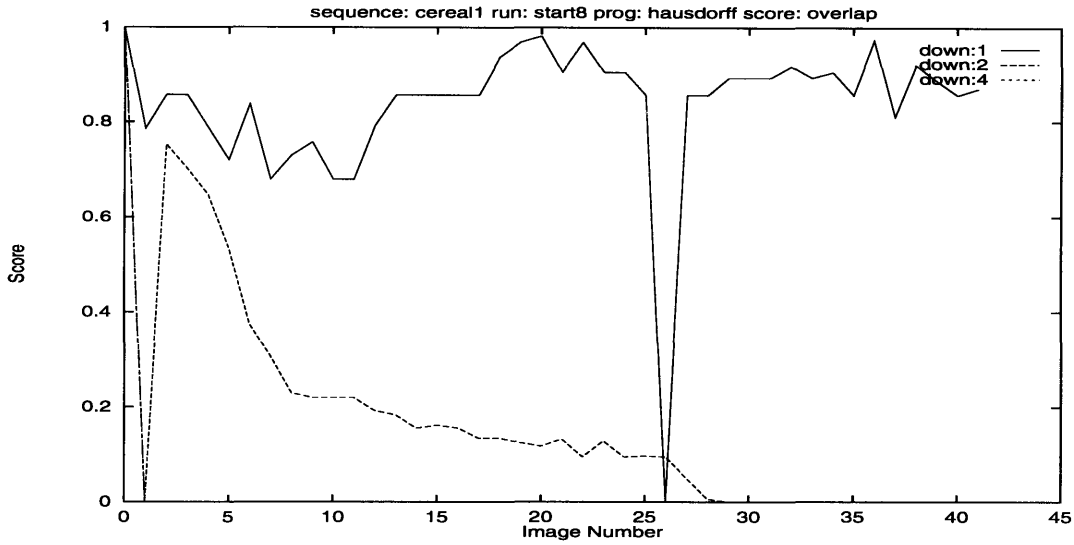


Figure 4-12: Scores plotted for the Hausdorff algorithm run on the cereal1 sequence at three resolutions, starting with frame 8 of each sequence.

good data points on what resolutions are necessary for this algorithm to perform well. Table 4.1 summarizes this and additional data on the effects of size on performance of the Hausdorff algorithm. With the exception of the takeh sequence, the algorithm was unable to track the objects at any size less than full resolution. In the takeh sequence the tracking was good at half resolution. This is presumably because the man occupies a much larger area in the field of view than the other objects. In fact, at half resolution his size is comparable with the objects in the can4 and cereal1 sequences at full resolution. Observe also that although the cereal1 box is smaller than the squares1 object that it has much more texture and hence many more edge pixels. This contributes to the algorithm's better performance.

Finally, we can compare the effect of a cluttered background on the Hausdorff algorithm by looking at the can4 and can5 sequences. We will only consider the full resolution runs as the algorithm was unable to track at the lower resolutions. Figure 4-13 compares the tracking on these two sequences at full resolution. The performance on the can4 sequence is good, except after frame 26 where it is unable to recover from the can having stopped momentarily. On the can5 sequence, the algorithm is able to track only up until frame 15, and even then with less than half the score. Single stepping the algorithm through the sequence, we see that the complex background has broken the model update step where the model bounding box is expanded or contracted. In the sequence, there is so much texture that even after removing the stationary background many edge fragments remain. This causes there to almost always be a significant number of edge pixels near the model boundary, and so the model expands to much larger than the can — more than double the can's size in each dimension. As the can moves past the left edge of the poster background into a region with no background texture, the model contains so much background clutter that it begins tracking the

Sequence	Resolution	Object Size	Model Pixels	Tracking
can4	full	53x28	22	good
	half	26x14		no
	quarter	13x7		no
cereal1	full	54x73	68	good
	half	27x36	10	poor
	quarter	13x18		no
squares1	full	85x88	34	good
	half	42x44	9	poor
	quarter	21x22		no
takeh	full	75x180	126	good
	half	37x90	36	good
	quarter	19x45		no

Table 4.1: Data showing which size objects were able to be tracked well by the Hausdorff algorithm on the start8 experimental runs. Model pixel counts are for the sparse model, using a model density factor of 0.10.

remaining background pixels instead. Figure 4-14 shows the edges detected in frame 20 of the can5 sequence as well as those that remain after background cleaning. There is definitely a large cloud of pixels remaining.

Since it is possible that part of the problem is that the can is on the verge of being too small to track, we will show more experiments later in Section 4.7 confirming that a complex background causes big problems with the model expanding into the background.

Thus, starting the experiments on the Hausdorff tracker with motion in the image helped a great deal and gave good tracking results. There are still problems caused by the object stopping its motion, especially when it comes to a stop gradually. The algorithm also has difficulty when the background is highly textured. Finally, we have obtained a rough lower bound of approximately 50x100 pixels on the object size necessary for the algorithm to successfully track an object.

4.3 Decreasing the Frame Rate

Our next set of experiments was designed to determine how a lower frame rate would affect each of our algorithms. We re-ran each of our previous experiments (both starting with frame zero and starting with frame eight of each sequence), sampling the images temporally by a factor of two in each case. Note that decreasing the frame rate is equivalent to increasing the speed of the moving objects. The results for each algorithm are given below.

4.3.1 Hausdorff Algorithm

The Hausdorff algorithm performed very similarly on the reduced frame rate experimental runs as it did on the full frame rate runs. The tracking results for the takeh sequence are shown in Figure 4-15 for the full sequence at half frame rate and each of the three sizes. Compared with the same runs

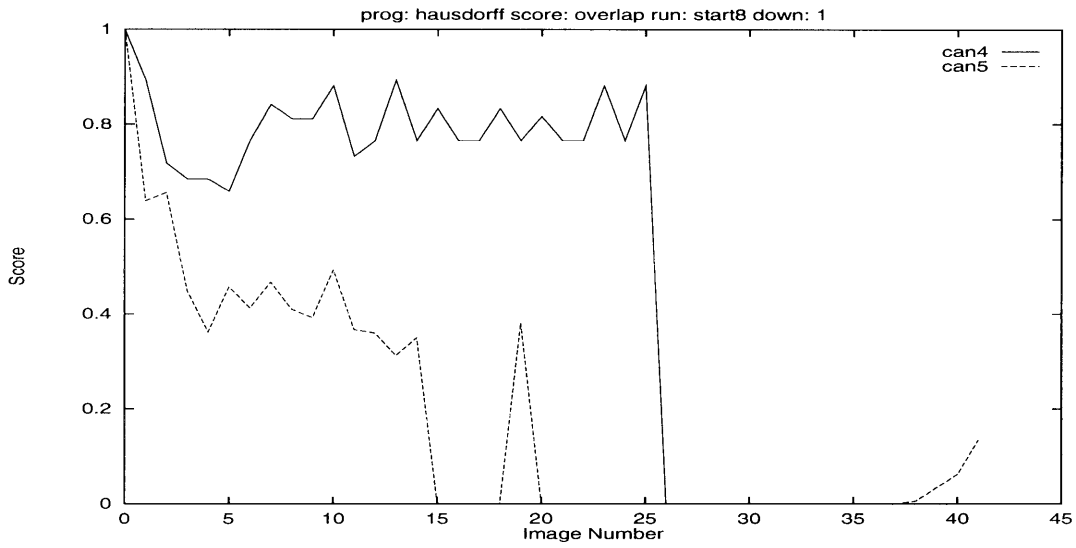


Figure 4-13: Comparison of the tracking on the can4 and can5 sequences by the Hausdorff algorithm, at full resolution

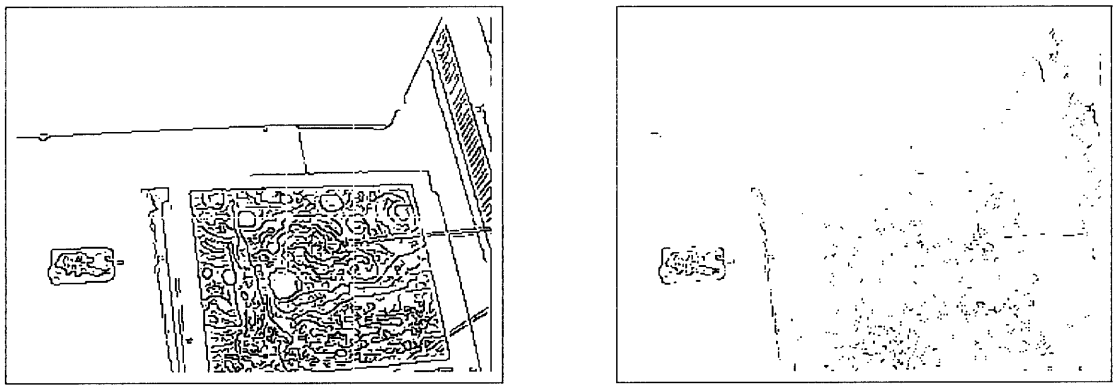


Figure 4-14: Left: edges detected in frame 20 of the can5 sequence. Right: cloud of edge pixels which remain after performing the Hausdorff algorithm's stationary background removal operation. It is this large cloud of pixels which confuse the model update and cause the model to grow far into the background.

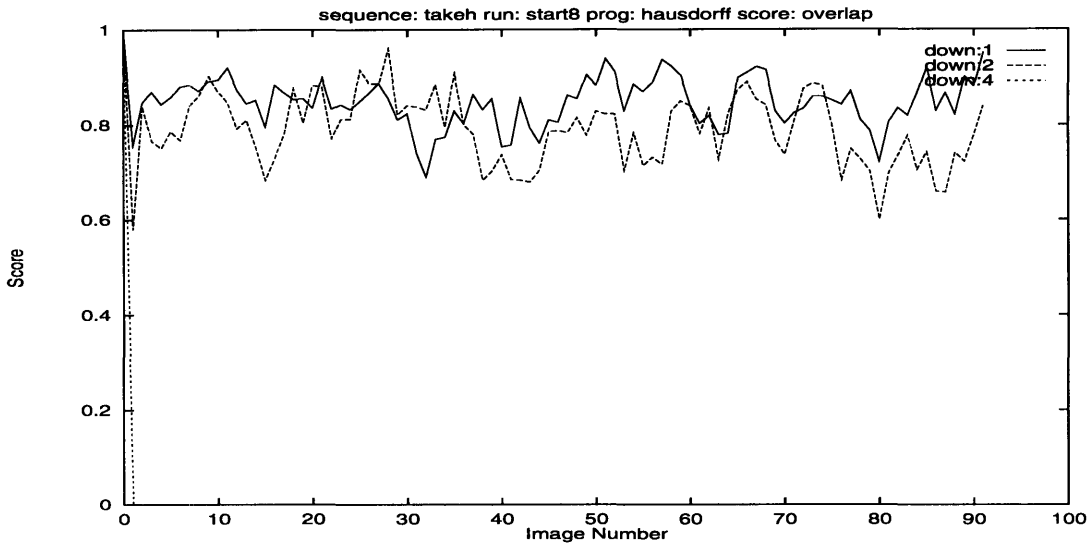


Figure 4-15: Results for the Hausdorff algorithm on the takeh sequence starting with frame 8 and at one half frame rate.

at full frame rate in Figure 4-16 we see that both the full and half resolution runs scored almost 0.1 higher for the full frame rate experiments. Still, the performance was quite good for the half frame rate runs, with an average score above 0.7 at both full and half resolutions. The algorithm was unable to track at quarter resolution in both sets of experiments. The tracking was just slightly worse on the controlled motion sequences at the lower frame rate too. However, the problems due to the sequences starting with no motion and the objects stopping momentarily when they changed directions were no longer a problem because at the slower frame rate the objects either did not stop or did not stop for enough frames to adversely effect the tracking. These results are as expected because the Hausdorff algorithm searches the whole image for a match with the model from the previous frame. Hence, faster motion should not be a problem as long as the appearance does not change significantly.

4.3.2 Correlation Feature Tracking Algorithm

The tracking observed for the correlation feature tracker at the reduced frame rate was nearly as good as that at full frame rate, and in some instances was actually better. For example, Figure 4-17 shows the results of the tracker on the cereal1 sequence at one half frame rate. The scores are almost identical to those for the full frame rate given in Figure 4-9 for full and half resolutions. The scores for the quarter resolution run are much better than those at the full frame rate, 0.80 versus around 0.60 at worst. Because we are only looking at every other image, the sub-pixel vertical motion at full frame rate is no longer always less than one half pixel and the algorithm does not drift vertically off of the box as much. This behavior is exhibited on the other controlled motion sequences as well. The tracking on the takeh sequence was nearly as good at half frame rate as

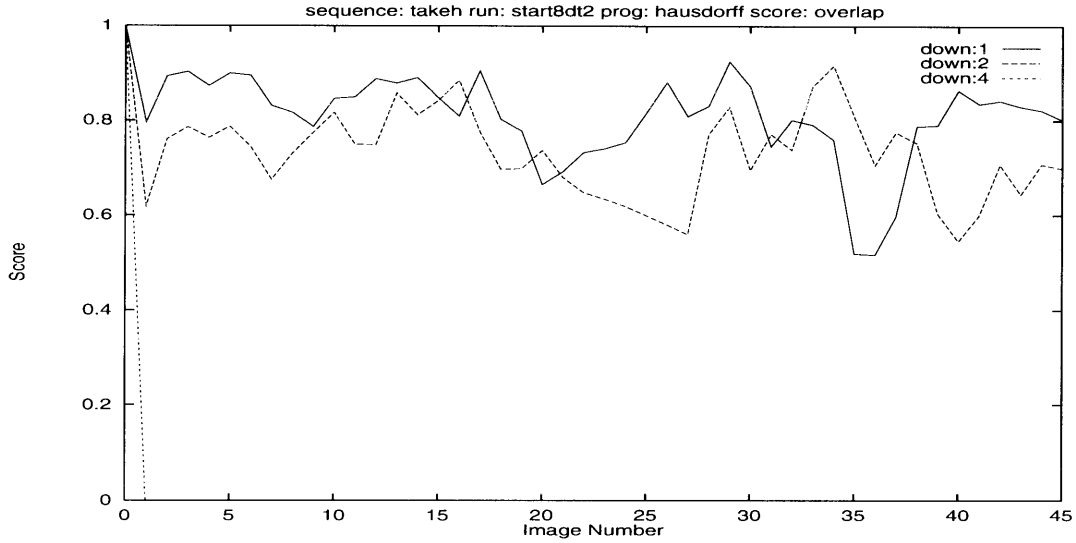


Figure 4-16: Results for the Hausdorff algorithm on the `takeh` sequence starting with frame 8 and at full frame rate, for comparison with Figure 4-15.

full frame rate, except at quarter resolution where it was somewhat worse. Overall, decreasing the frame rate by a factor of two did not greatly effect the correlation feature tracker. This result is not terribly surprising. So long as the predictions given for the new feature location are accurate to within the search radius R_S and the appearance of the feature does not change significantly, a faster moving object should not cause serious problems.

4.3.3 Motion Based Algorithm

Decreasing the temporal sampling rate did not significantly effect the motion based algorithm either. Unlike the results for the other two algorithms, this result is surprising because of the limited search radius in the optical flow computation. We saw in the initial experiments of Section 4.1 that the tracking quality deteriorated as the inter-frame motion exceeded the search radius. Since we had just barely met this condition above, one of the primary reasons for the decreased temporal sampling rate experiments was to exceed the search radius by a larger margin and cause more pronounced problems, especially at the higher spatial resolutions which were unaffected in the previous experiments.

Figure 4-18 shows the results for the motion based algorithms on the full `cereal1` sequence, at one half frame rate. The inter-frame displacements of the box peak at 24 pixels per frame between frames 8 and 9 and frames 9 and 10, for the full resolution images. Recall that the optical flow search radius is only 12 pixels at this spatial resolution. As the images are sampled spatially, both the inter-frame displacements and the search radius decrease linearly so at all resolutions the motion peaks at double the search radius. Hence, we expect the optical flow to fail over a range of image frames centered at frame 10 due to no matches being found for the optical flow patches. However, the tracking is very good at all three spatial resolutions with the exception of frame 10 of the half

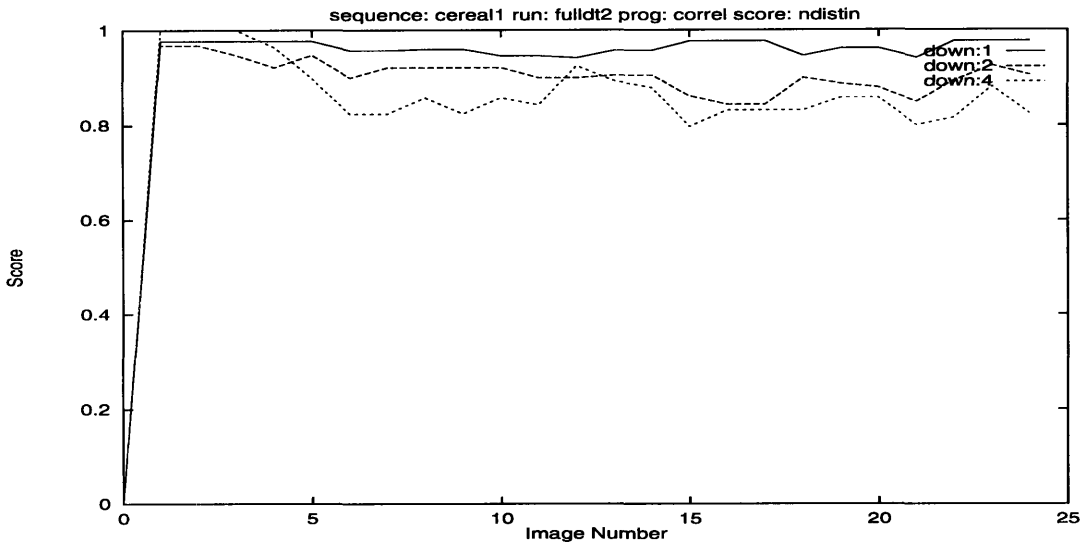


Figure 4-17: Results for the correlation based feature tracking algorithm on the full `cereal1` sequence, at one half frame rate.

resolution run. But looking at the motion pixels detected in this frame, shown in Figure 4-19, we see that in fact motion has been detected over most of the cereal box's location. The motion pixels detected, however, have been grouped into several connected components, and the algorithm chose a very small component containing only 3 pixels over the correct one containing almost 400 since the small component was located closer to the previous box position. These few pixels were due to motion detected at the right edge of the box's old position.

Why is motion being detected even though the search radius should be much too small? We did not stumble upon the answer to this question until analyzing the experiments of Section 4.6. The answer will be explained in that section since the reason is more easily demonstrated with those experiments. We will hint at the reason with a comparison of the results of the runs on the `can4` and `can5` sequences. Figure 4-20 shows the results of this comparison at full spatial resolution. Recall that the only difference between these two sequences is that the flat white background in `can4` has been replaced by a highly textured and brightly colored background in the `can5` sequence. We see that the algorithm tracks well on the `can4` sequence but that on the `can5` sequence the score drops to zero from frame 9 through frame 12 as we had expected in all the sequences. Clearly the change in background has played a role here.

4.4 Decimation versus Downsampling

The operation of reducing the sampling rate of a discretely sampled signal is called downsampling or compression. For a one-dimensional discrete signal, $x[n]$, a downsampled signal with sampling

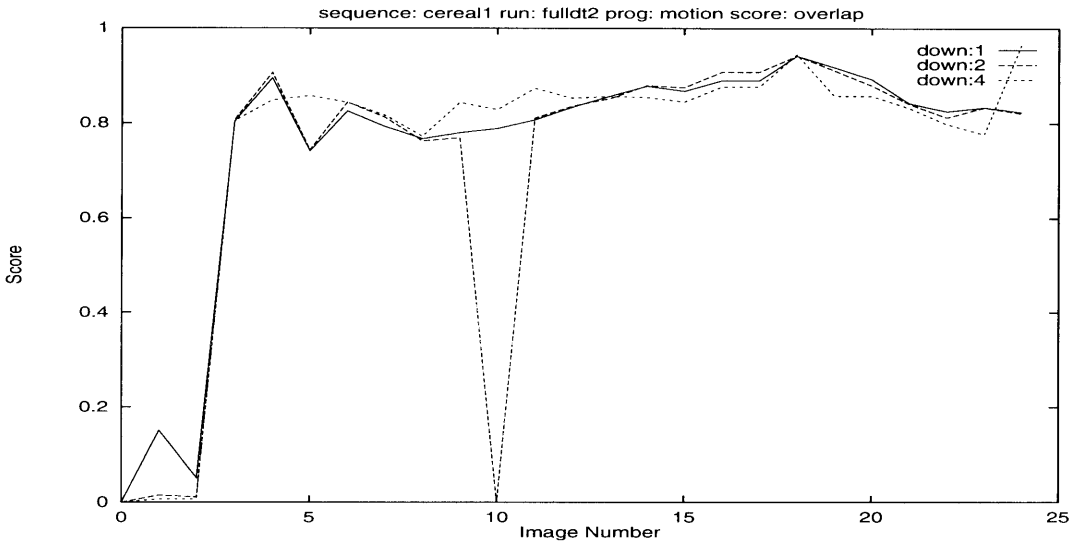


Figure 4-18: Results for the motion based algorithm on the full `cereal1` sequence, at one half frame rate.

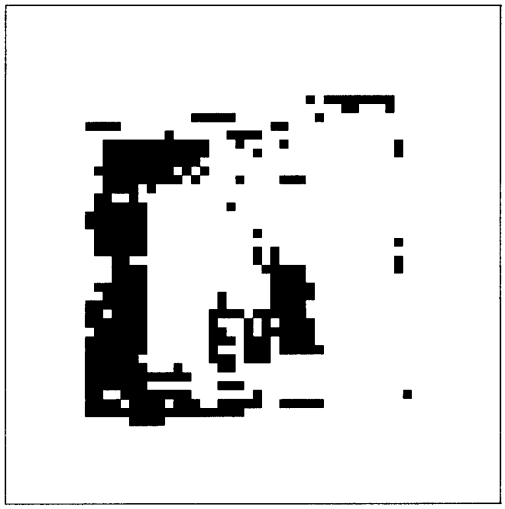


Figure 4-19: Closeup of the motion detected in the vicinity of the cereal box in frame 20 of the half resolution run of the motion based algorithm on the full `cereal1` sequence, at one half frame rate.

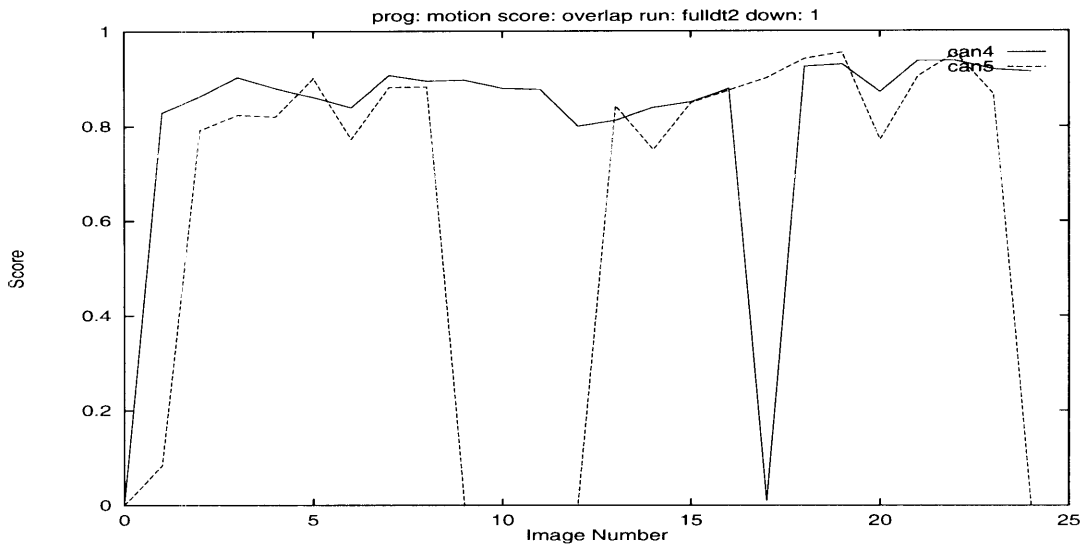


Figure 4-20: Results for the motion based algorithm on the full can4 and can5 sequences, at one half frame rate and at full spatial resolution.

rate reduced by a factor of M can be obtained by defining a new sequence

$$x_d[n] = x[nM]. \tag{4.2}$$

This is in fact how we have obtained the reduced size images for the half and quarter spatial resolution experiments in Equation 4.1. These equations are simply the two-dimensional analogues of Equation 4.2, with M equal to two and four for the half and quarter resolutions, respectively. Whenever a discretely sampled signal is downsampled, aliasing can occur. Aliasing takes place when the high frequency components of the original signal can no longer be represented at the lower sampling rate. So far we have ignored this problem in our experiments and simply downsampled the images to yield the half and quarter resolution images. In this section we will compare the results of all the experiments we have done so far using simple downsampling with results obtained by doing the the proper pre-filtering to avoid aliasing followed by downsampling, a process referred to as *decimation.*

4.4.1 Aliasing

A discrete signal $x[n]$ can be represented as the superposition of complex sinusoids by the Fourier integral (see [18])

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} x[n] e^{j\omega n} d\omega, \tag{4.3}$$

where the Fourier transform, $X(e^{j\omega})$ is given by

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}. \tag{4.4}$$

The Fourier transform allows us to think of the signal in the "frequency domain" by considering what happens to the different frequency sinusoids as the signal is processed. For discretely sampled signals, only the frequencies ω in an interval of length 2π need be considered since for any sinusoid of frequency ω_0 it is easy to see that all sinusoids of frequencies $\omega_0 + 2\pi r$ are equivalent:

$$\begin{aligned} x[n] &= A cos[(\omega_0 + 2\pi r)n + \phi] \\ &= A cos[\omega_0 n + 2\pi r n + \phi] \\ &= A cos(\omega_0 n + \phi) \end{aligned}$$

We have chosen to use the frequency range $-\pi < \omega_0 \leq \pi$. Because of the limited frequency range that can be represented with discrete signals, and because downsampling by a factor M increases the frequencies in the signal by a factor M, any Fourier components with frequencies outside the range $\frac{-\pi}{M} < \omega \leq \frac{\pi}{M}$ will exceed the limited frequency range and wrap back into the $-\pi < \omega_0 \leq \pi$ range. This phenomenon is called aliasing and it is undesirable because the frequency components that get aliased (wrap back) introduce new frequency components which did not previously exist.

Consider for example the signal $x[n] = cos(\pi n)$ of frequency $\omega_0 = \pi$. Downsampling by a factor of two gives

$$\begin{aligned} x_d[n] &= cos(\pi(2n)) \\ &= cos(2\pi n) \\ &= 1 \end{aligned}$$

We have gone from having a high frequency signal to having a constant (zero frequency) signal. Even worse, if we had sampled the odd pixels instead of the even pixels, we would have gotten a different result:

$$\begin{aligned} x_d[n] &= cos(\pi(2n+1)) \\ &= cos(2\pi n + \pi) \\ &= -1. \end{aligned}$$

Clearly we should avoid aliasing when downsampling our images.

4.4.2 Decimation

Aliasing can be avoided by pre-filtering the discrete signal to remove any frequency components that would alias. The operation of pre-filtering followed by our usual downsampling is called 'decimation'. This is accomplished by a low-pass filter which passes frequencies from $\frac{-\pi}{M}$ to $\frac{\pi}{M}$ unattenuated and completely attenuates frequencies outside this range. Unfortunately this ideal low-pass filter with cut-frequency $\frac{\pi}{M}$ is not realizable in a discrete system and must be approximated.

One simple way to design an approximating filter is to use the Kaiser window method (see [18] and [12]). This method designs a finite impulse response (FIR) filter from the infinite impulse response of the ideal filter by multiplying the ideal response by a window of finite extent. The ideal impulse response for a low pass filter with cut-frequency ω_c is given by

$$h_{lp}[n] = \frac{sin(\omega_c n)}{\pi n}. \tag{4.5}$$

Using the window method, we design our approximation $h'_{lp}[n]$ as

$$h'_{lp}[n] = h_{lp}[n]w[n] \tag{4.6}$$

where the window $w[n]$ is zero for all n outside $-N \leq n \leq N$. Hence, h'_{lp} has finite extent and the filter can easily be implemented by convolution.

In our experiments we have used a window of size 27 ($N = 13$) to design one-dimensional low pass filters for the cut frequencies $\omega_c = \frac{\pi}{2}$ and $\omega_c = \frac{\pi}{4}$ which are needed to decimate by factors of two and four. These filters approximate the desired response much better than should actually be necessary. We can then approximate the appropriate two dimensional ideal low-pass filter by applying the one dimensional filter twice in succession: once oriented horizontally and once vertically. We have effectively designed a two dimensional separable filter (see [12]). Hence, to produce our decimated reduced resolution images we first apply the appropriate anti-aliasing filter to each image in a sequence and then downsample the resulting image as we had done previously. The results of the experiments comparing decimation with downsampling alone are given below for the three algorithms.

4.4.3 Hausdorff Algorithm

The only experimental run we have conducted in which the Hausdorff algorithm performed well at less than full resolution was on the `takeh` sequence, on the `start8` runs, at half resolution. Figure 4-21 shows a comparison of the tracking on this run for both decimation and downsampling by a factor of two. It is evident that pre-filtering to avoid aliasing had no appreciable effect. At some times the tracking is slightly better and at others slightly worse.

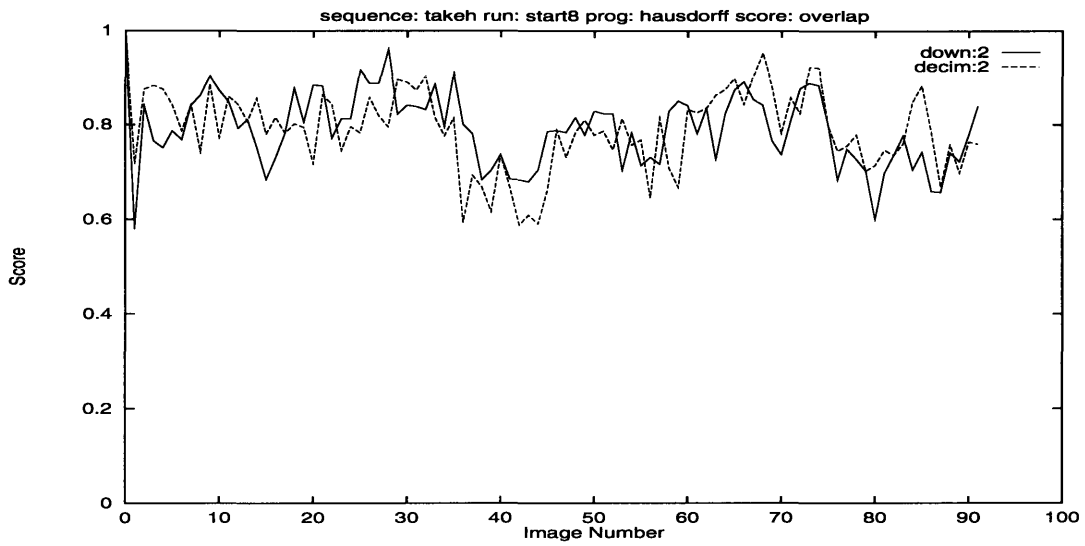


Figure 4-21: Results for the Hausdorff algorithm on the takeh sequence, at half resolution and starting with frame eight, comparing decimation to downsampling.

4.4.4 Motion Based Algorithm

There were no clear differences for the experimental runs comparing decimation and downsampling using the motion based algorithm either. The tracking for the cereal1 sequence was typical of these experiments even though we would have expected any difference to be on this sequence due to the high-frequency texture on the cereal box. Figure 4-22 shows the results for the quarter resolution experiment. The tracking is nearly identical for both decimation and downsampling. Over all the experiments run, there were definitely minor differences for this algorithm, however neither sampling method was better overall than the other. Rather, it seems each perturbed the image data in slightly different ways such that in cases where the algorithm was close to breaking, one sampling method may have pushed it over the edge while the other did not.

4.4.5 Correlation Feature Tracking Algorithm

The correlation based feature tracker was likewise imperceptibly affected by the difference between decimation and downsampling. Figure 4-23 shows the results for the squares1 sequence at both half and quarter resolution. At half resolution, both are identical through frame 15 but then decimation does significantly better for the remainder. The only difference was that in frame 16 the algorithm chose a vertical displacement of two pixels using the decimated images and only one for the downsampled images. As with the other algorithms, we believe that this is simply the result of a chance perturbation of the data, not a result of decimation being better. The fact that the two runs diverge however points to the fact that the algorithm does not have the property that small perturbations of the initial conditions will eventually converge to the same result. At quarter resolution the two

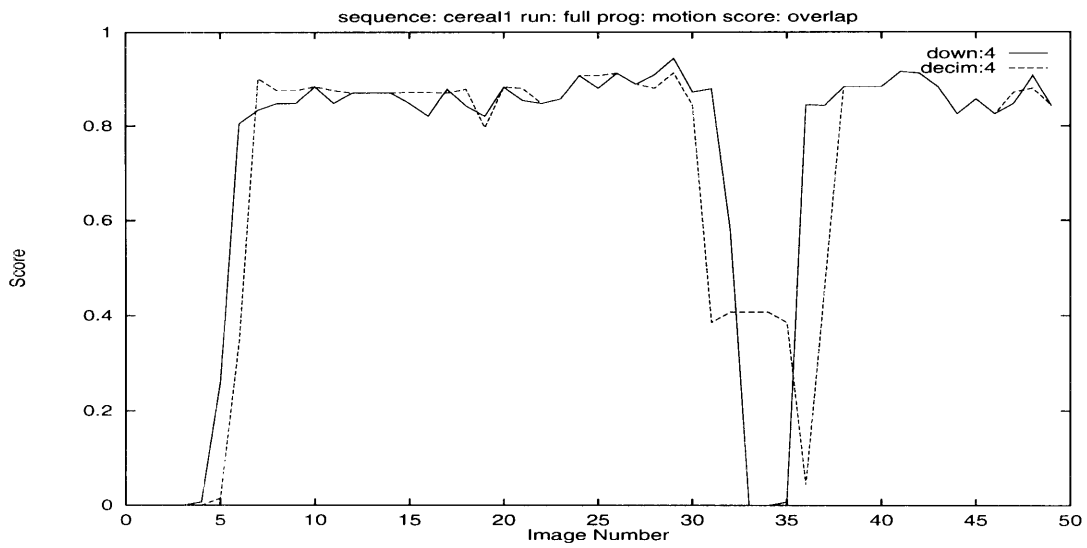


Figure 4-22: Results for the motion based algorithm on the full **takeh** sequence, at quarter resolution, comparing decimation to downsampling.

sampling methods give identical results.

Hence, we have seen that for all three algorithms there was no difference between whether an anti-aliasing filter is applied before sampling the images spatially. This is good because this is an expensive operation computationally, though it would surely suffice to use a less optimal filter than we did. In our experiments, pre-smoothing took approximately 4.4 seconds per frame on a SPARCStation 5 for the RGB images.

4.5 Varying the Hausdorff Model Density

We conjectured in Section 4.2 above that we may be able to reduce the effect of no motion on the Hausdorff algorithm as well as enable it to track smaller objects better by increasing the model density factor, σ_M. Recall that the model density specifies a fraction of the new pixels selected by the model update equation (Equation 2.15) to use as the new model. This step of the model update was added to reduce the effect of background edge fragments which remain after the background removal step. As this parameter is set lower, chances get better that a small number of background pixels which get incorporated into the model M_t at time t do not get re-selected to be placed into the next model M_{t+1}. It has the added benefit of speeding the computation.

All previous experimental runs of the Hausdorff algorithm were done using a model density of 0.10. For the experiments of this section, we re-ran the algorithm on each of the image sequences at full resolution as well as the **takeh** sequence at half resolution, using each of the following model densities: 0.20, 0.40, 0.60, 0.80, and 1.00. For these experiments, we started each run with frame 8 of the image sequence, as we had for those in Section 4.2.

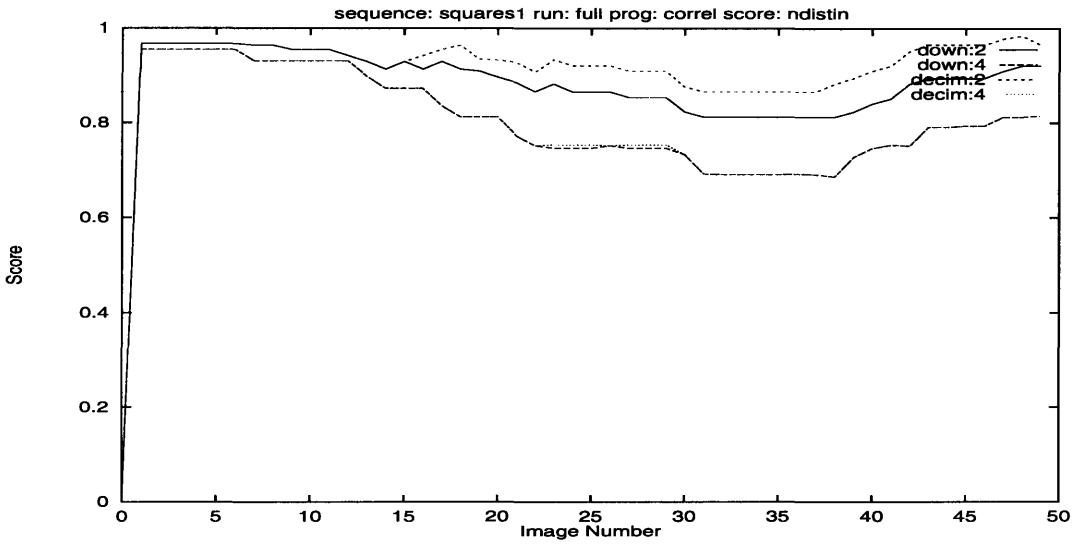


Figure 4-23: Results for the correlation based feature tracking algorithm on the full `squares1` sequence, at both half and quarter resolution, comparing decimation to downsampling.

Our results show that as the model density is increased the algorithm does handle cases where the object stops moving briefly better, but that the problems described in Section 4.2 in which the model expands to contain a significant amount of background pixels become much worse. Figure 4-24 shows a comparison of the tracking on the `cereal1` sequence for a range of model density values. We had argued in Section 4.2 that the fact that the algorithm had recovered from the box stopping in frame 26 was purely by chance, considering that a model containing only 7 pixels was matched in frame 27 to continue the tracking. With the model density set at 0.20, we find even better evidence for this claim. On this run, the new model generated in frame 26 had 12 pixels. In frame 27, however, it happened to match best to some remaining background edge fragments in the upper right corner of the image. Somehow, the model generated there then matched part of the cereal box in frame 28 and resumed tracking the box. By the time the model density is increased to 0.60 (and greater), however, the model never dwindles to less than 50 pixels and the tracking is good through the stoppage of motion.

The downside to increasing the model density is seen in frames 5 through 20. With each increase in the model density, the tracking gets progressively worse in frames 5 to 15. The tracking improves greatly for all the values of the model density beginning at frame 15, however, and by frame 20 all the runs are scoring better than 0.90. This odd behavior is due to a lack of any background texture in the vicinity of the box in these frames. Looking at the tracking for the `squares1` sequence shown in Figure 4-25 we see that this does not happen there. Instead the model continues to grow larger and larger as time progresses.

We can see that the effect of increasing the model density is negligible by studying Figure 4-26. The figure plots the overall score for each sequence as the model density is increased from 0.10 to

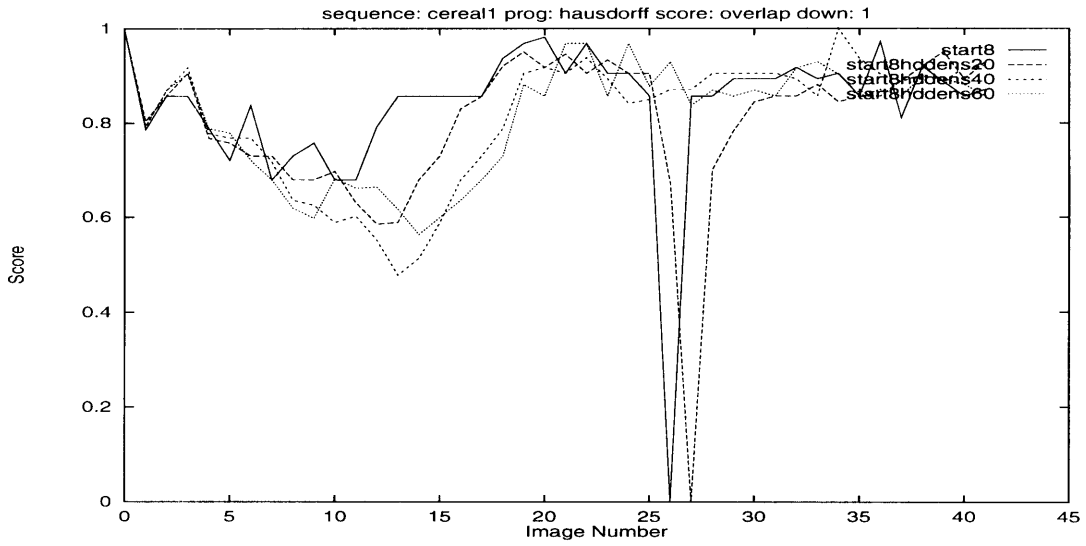


Figure 4-24: Results for the Hausdorff algorithm on the full resolution `cereal1` sequence, starting with frame 8, and run for the following values of the model density: 0.10, 0.20, 0.40, and 0.60.

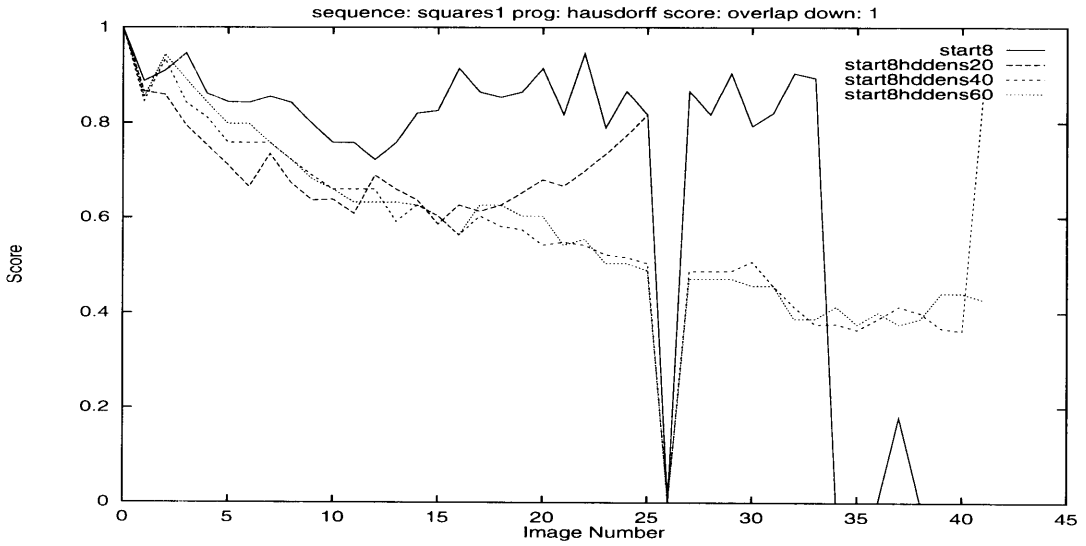


Figure 4-25: Results for the Hausdorff algorithm on the full resolution `squares1` sequence, starting with frame 8, and run for the following values of the model density: 0.10, 0.20, 0.40, and 0.60.

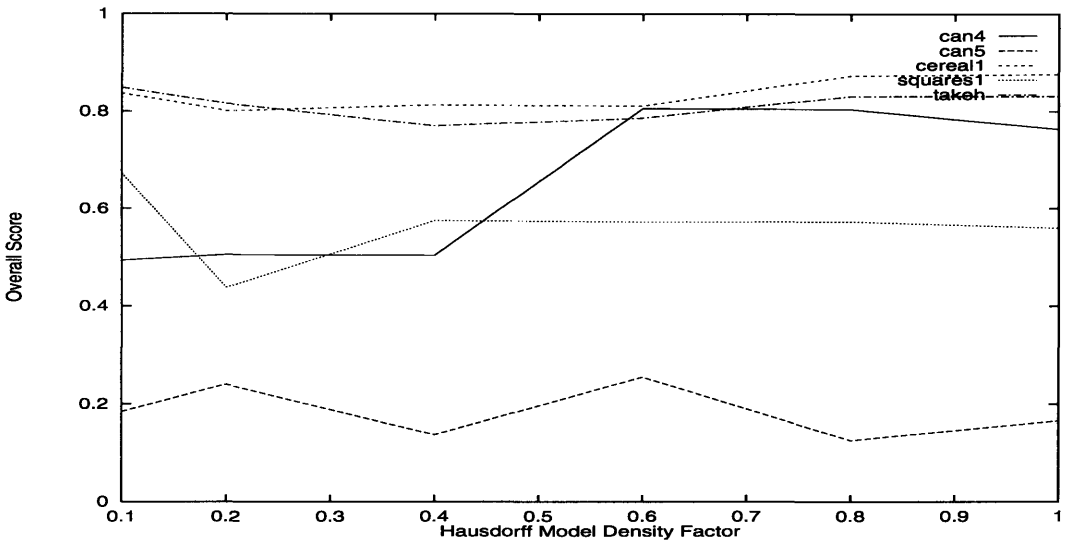


Figure 4-26: Overall results for the Hausdorff algorithm on each sequence as the model density is varied from 0.10 to 1.00.

1.00. The curves for all sequences except the can4 sequence are nearly flat, showing no benefit from increasing the model density. In fact, the upward trend exhibited by the scores on the can4 sequence as the model density is increased is misleading. At densities greater than 0.60, the algorithm recovered from the can stopping in frame 26. However, at each of those densities, the model from frame 26 matched some background pixels in frame 27 and then the model from frame 27 somehow locked back onto the can. The trend is really not valid since this does not really constitute doing "better".

Hence, we have seen that increasing the Hausdorff algorithm's model density can alleviate problems due to the object stopping briefly, however, this improvement is offset by the algorithm increasingly expanding the model to include background. It seems as though there is no good value which solves both problems satisfactorily.

4.6 Randomly Permuting the Sequences

Our next set of experiments involved randomly permuting the ordering of the image frames in each sequence and running the algorithms on the resulting sequences. This was initially done to emphasize that the Hausdorff tracker searches everywhere in the image for a match with the model from the previous frame, whereas the other two algorithms only perform local searches. Even though we expected this to work for the Hausdorff algorithm only for the controlled motion sequences in which there is no significant shape change across the whole sequence, we ran the experiments on all five sequences and at the three resolutions we have so far been considering. In addition to what we had set out to show, these experiments uncovered other interesting insights into all three algorithms.

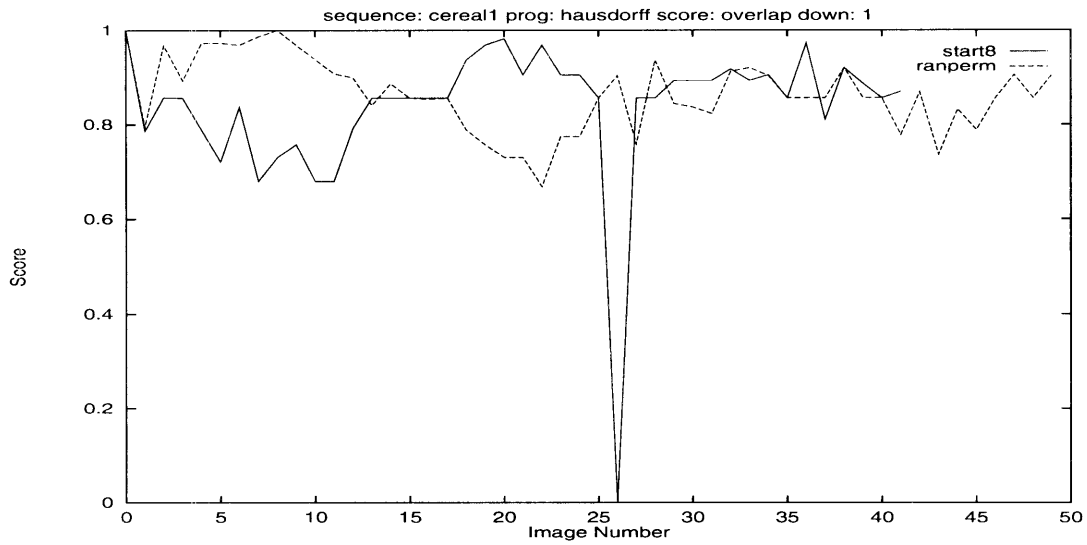


Figure 4-27: Results for the Hausdorff algorithm on the full resolution `cereal1` sequence. Results are shown for the sequence starting with frame 8 as discussed in Section 4.2, and for a randomly permuted frame sequence.

4.6.1 Hausdorff Algorithm

As we expected, the Hausdorff algorithm tracked well on the randomly permuted controlled motion sequences. Figure 4-27 shows the result of running the algorithm on the randomly permuted `cereal1` frames, compared against a sequential run starting with frame 8, both at full resolution. The tracking is of the same quality for the permuted run as for the sequential image ordering. With the permuted sequence, the box never stops moving, however, so the problems we have discussed due to this were not encountered.

What we did not expect was that the algorithm would track well on the permuted `takeh` sequence. The man in this sequence undergoes significant shape change through the course of the sequence and we do not expect that the shape in an early frame should match that in a much later frame where his appearance is very different. Figure 4-28 shows, however, that this is exactly what happened. The tracking for the permuted sequence is not as good as for the sequential ordering, but it scored only about 0.10 less on average, and is clearly quite good. There are only two frames in which the object has been declared 'not found' by the algorithm, and the tracking has quickly resumed in the following frame. In fact, the only reason the tracking scores are lower is that the man has typically changed size (due to depth change) enough between permuted frames that the bounding box of the model from the previous frame differs significantly in size. It is not due to a matching problem, as the match scores average about 4.0 pixels in the permuted run and 3.0 in the normal run. There is significant overlap in the distributions which makes this difference insignificant. See Figure 4-29 for a plot of these distributions.

Looking more closely at the matching process, we see that models of the man are being matched

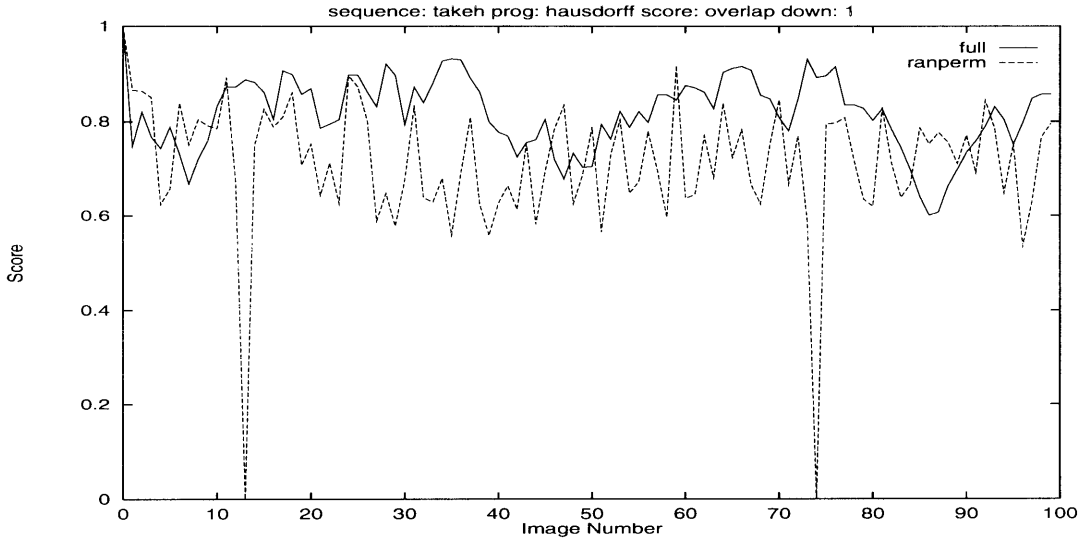


Figure 4-28: Results for the Hausdorff algorithm on the full resolution takeh sequence. Results are shown for the full sequence in sequential order and for a randomly permuted frame sequence. The overall scores are 0.816 for the normal sequence and 0.712 for the permuted sequence.

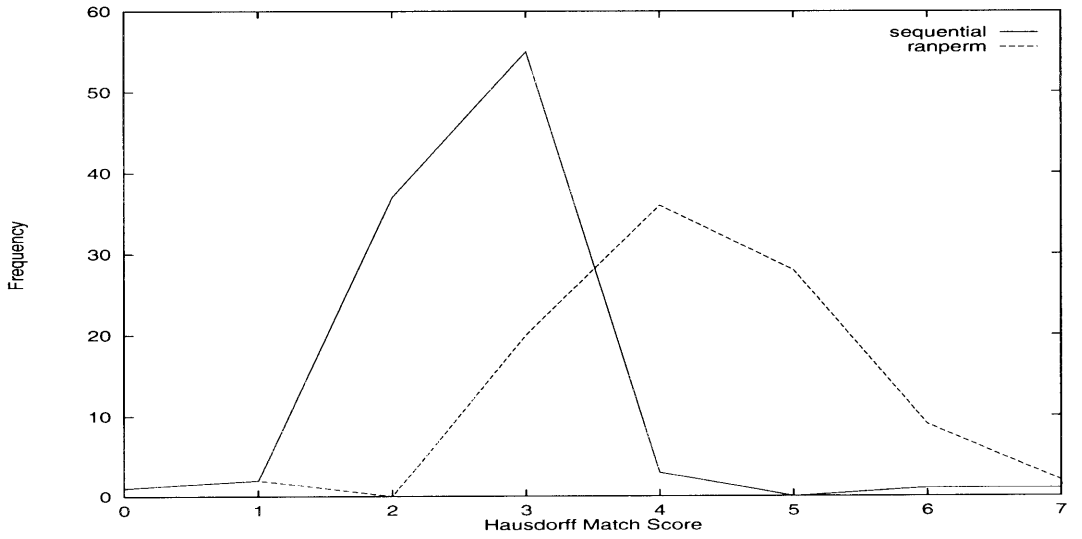


Figure 4-29: Histograms of the Hausdorff algorithm match scores (the value of Equation 2.14 at the match location) for the takeh sequence, on both the sequential run and the randomly permuted run. There is not a significant difference in these distributions.

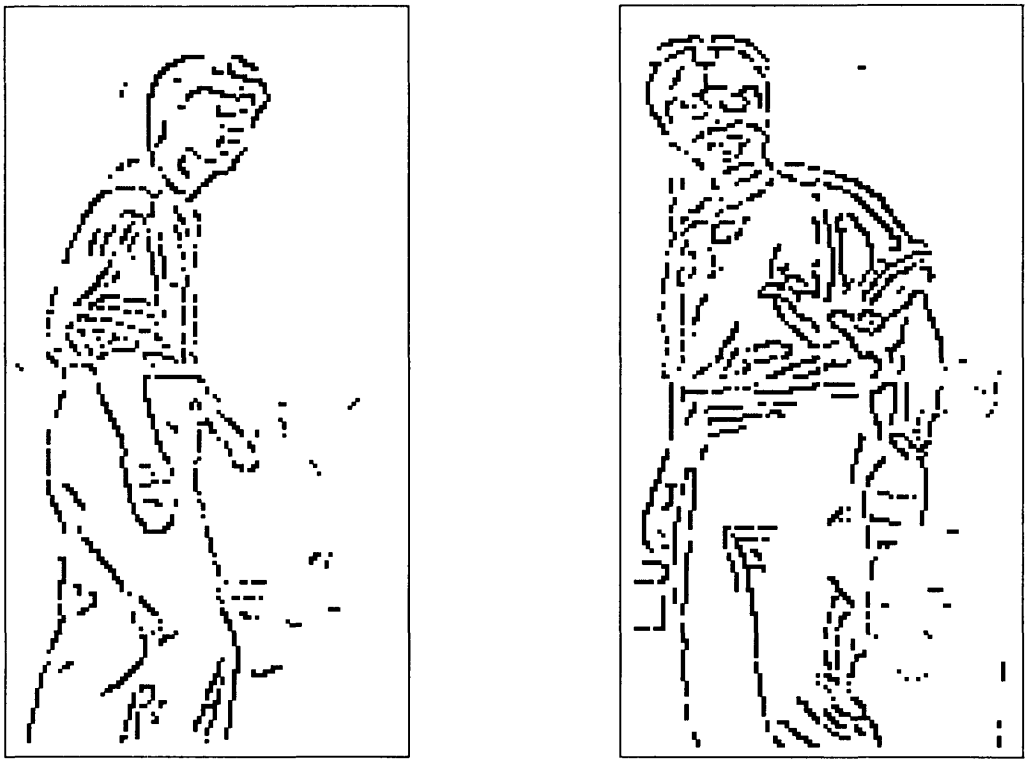


Figure 4-30: Example Hausdorff match for the randomly permuted run on the takeh sequence. The left image is the model from frame 10, M_{10}. The right image shows a closeup of the portion of image 11 which was matched by the model. Note that the model shown is the complete model; the model actually used in the matching was a sparse model containing about ten percent of these pixels. The match score was 4.0 pixels.

to images in which his appearance is very different. Figure 4-30 shows the match of the model M_{10} from image 10 to image 11 (frames 89 and 31, respectively from the in-order sequence). The man is facing completely different directions in these two frames and the 2D appearance is very dissimilar. There is no way that the majority of the features matching actually correspond to the same 3D features. These results bring the shape discrimination ability of this algorithm into question. Is the algorithm doing anything more than detecting an oval shaped cloud of edge pixels? We had hoped that the shape matching used by the algorithm might help solve the person constancy problem for tracking multiple people, but this now seems doubtful. Further experiments will be performed in Section 4.8 to determine whether this will be possible.

4.6.2 Motion Based Algorithm

Running the motion based algorithm on the permuted sequences, we expect the tracking to be very poor because the object's displacement is almost always far enough to be outside the search radius of the optical flow. However, Figure 4-31 shows that the tracking is actually quite good. There are long periods of 30 or more frames in which the tracking score remains above 0.5. This result is quite

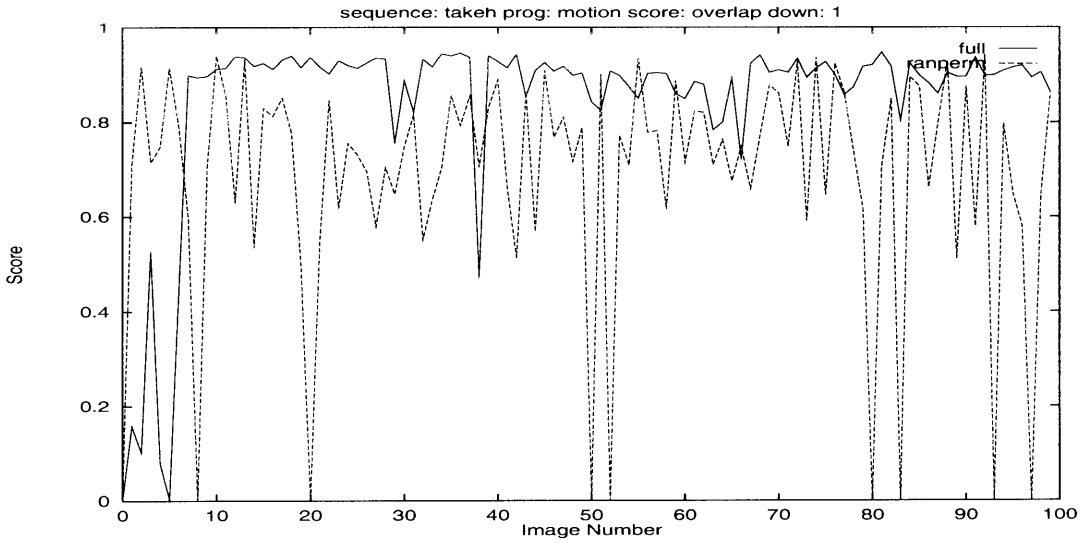


Figure 4-31: Results for the motion based tracker on the full resolution takeh sequence. Results are shown for the full sequence in sequential order and for a randomly permuted frame sequence. The overall scores are 0.846 for the normal sequence and 0.688 for the permuted sequence.

unexpected.

Figure 4-32 shows the motion detected during the run at half resolution. The man has moved from the left position to the right between frames 40 and 41. There is a difference in intensity detected at both locations, but motion is for the most part only detected around the perimeter of the current position to the right. Examining Equation 2.10 carefully, we see that motion is detected at pixel locations where the optical flow from the previous image to the current image was non-zero. Hence, we see that in Figure 4-32 the primary motion being detected is that of the background which had been in places now occupied by the man. This background is flowing outward, matching other background around the outline of the man's position that was not detected to have changed. Motion is not detected in the interior of the man's new position because the optical flow search radius is small enough that the search from these regions does not consider any background locations in the current frame.

Recall in Section 4.3 that we were surprised by the motion based algorithm working well at half frame rate even though the inter-frame motions were in many cases much larger than the optical flow search radius. We hinted that the anomaly had something to do with the background, as it occurred in the can4 sequence but not in the can5 sequence which has a highly textured background. We can now understand this result: with the complex background, the background covered by the can's new position found no match at any nearby position and no flow was detected. However, with a flat background, the occluded background matched any other background nearby and flow was detected.

Hence we have seen that the optical flow computation and motion detection gets confused by

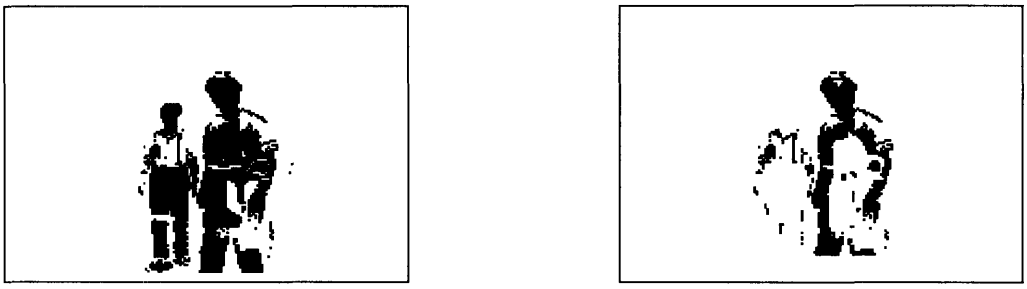


Figure 4-32: Example showing the motion detected by the motion based algorithm for frame 41 of the randomly permuted run on the **takeh** sequence. The left image is the difference image and the right image shows the motion detected (a subset of the difference). The man had been in the position to the left in frame 41, and moved to the position to the right in frame 41.

Activity	Approximate Speed	
	pixels/frame	cm/sec
walking (body)	3.0	80
running (body)	11.0	300
hand gesture (hand)	4.5	122
jumping jacks (hand)	19.1	520
jumping jacks (head)	6.1	165

Table 4.2: Data on the speeds at which humans can move. The results are for a resolution of 320x240 pixels using one of the camera systems comprised of a Chinon NTSC camera and 3mm lenses we have installed in the Intelligent Room. The data were taken from a depth of approximately 340 cm, an average depth expected within the room. The frame rate was 15 frames per second.

fast moving objects, especially on a flat background. It considers background to have moved out of the way of the object, to positions at which the difference image detected no change. Thankfully, this behavior should not be a serious problem if we sample at a high enough frame rate because humans are limited in how fast they can move. However it does bring into question the usefulness of the optical flow step in this algorithm. We will revisit this question in Section 4.11.

We have done some simple measurements to determine the speeds at which humans can move when performing various activities. Table 4.2 summarizes these results. The results show that at the full resolution of our captured image sequences, we expect humans not to be able to move faster than about 11 pixels per frame at a frame rate of 15 frames per second. This is within the optical flow search radius we have used in the motion based algorithm and hence we are not terribly worried about the problems described above.

4.6.3 Correlation Feature Tracking Algorithm

As expected, the correlation feature tracker was completely unable to track the objects in the permuted sequences. As soon as the object moves beyond the horizontal extent of the windows used to compute the predicted new location, the prediction is not within the search radius of the new location and the limited search does not find a good match. Since the algorithm does not have any

means to decide that it has not found a good enough match, it chooses the 'best' match with the background and starts tracking that. Of course the background is stationary and so through the rest of the sequence it remains in that fixed location. Using a match threshold on the match score (minimum SSD value) would help greatly in this case. Although we do not expect this situation in practice there will invariably be times when the prediction is bad enough that the feature is not found. Of course with any threshold there will be false positives and false negatives, but some value should work most of the time. Some other algorithm could then be used when the feature is lost to try to recover.

Despite the poor tracking, the motion difference computation gave very good detection of the object on the permuted sequences, especially when the object had moved far enough between frames to not overlap the previous position at all. Had we made the prediction windows as wide as the whole image the tracking would have been very good, however, this would preclude tracking more than a single person. If there were multiple moving people, the motion difference would detect all of them and it would not longer be the case that the center of mass of the motion difference pixels would give a good estimate of the object position. The fact that the motion difference requires no overlap in the objects position over three consecutive images to give accurate estimates would cause another problem as the number of moving people is increased. With more people it becomes less and less likely that their positions will not overlap for three frames, and the estimates will suffer.

4.7 Two New Sequences

In order to investigate the ability of the Hausdorff algorithm to distinguish between people with the aim of tracking multiple people, we next captured two new image sequences, the `kazu` and `magda` sequences described in Section 3.1. Before running our experiments on the discrimination ability of the Hausdorff tracker, we first ran the majority of our previous experiments up to this point on these sequences. We describe the results below, then in Section 4.8 proceed with the intended experiments on the Hausdorff algorithm.

4.7.1 Correlation Feature Tracking Algorithm

The correlation feature tracking algorithm did quite poorly on the two new sequences, particularly the `kazu` sequence. The feature patch was tracking the man's waist, but then as he turned sideways, his swinging arms caused the feature being tracked to change appearance and the tracker drifted downwards. The large amount of non-rigid motion exhibited by the man's walking legs, however could not be tracked and the patch quickly drifted off of his body and onto the stationary background by frame 13. In the `takeh` sequence the motion had been mostly such that the man was always facing the camera so this was not so much of a problem. The tracker clearly has trouble tracking

very non-rigid motions, especially when there are occlusions involved. In these cases the tracker quickly drifts from the original feature being tracked, often onto the background.

Initializing the tracker to track the man's head, there was no problem tracking it through the sequence, so long as the template size was made smaller than the head. When larger, as the tracking drifted slightly, enough background was included in the feature patch that it began tracking background instead of the man's head. Hence, choosing the template size is quite tricky. It must have a large enough support to match only the feature being tracked well, but not so big that it includes background, even if it drifts a small amount. It also needs to be small enough that the motion of the patch is roughly rigid. Otherwise there will be a large amount of drift and within a short time the tracker will no longer be tracking the original feature. Preventing the feature template from changing appearance too quickly as suggested by Wessler [24] may help in this case. He suggests updating the template at each step by averaging the previous template with the new feature patch. Having a match threshold in the algorithm would also help. If the match score (minimum SSD value) is above some threshold, then the algorithm would consider the feature to have either changed appearance too much or moved outside the search region. Some other technique would then be needed for recovering from this condition.

4.7.2 Hausdorff Algorithm

The Hausdorff algorithm performed reasonably well on the two new sequences, though not as well as it had on the `takeh` sequence. Because of the complex background in these sequences, the problems encountered previously in Section 4.5 with larger model densities were quite pervasive at the standard model density of 0.10. The algorithm was able to track the people at both full and half resolution, however, the models expanded to include a significant portion of the background. The overall scores were around 0.6 on these sequences, run at full resolution, due to the background being included in the models. Figure 4-33 shows the results of the models expanding into the background noise for image 15 of the `kazu` sequence. The bounding box has expanded to be about twice the width of the walking man. Otherwise the tracking was good on these new sequences.

4.7.3 Motion Based Algorithm

The motion based algorithm performed very well on the two new sequences. The tracking was comparable to that on the `takeh` sequence. No new information was gained from these experiments.

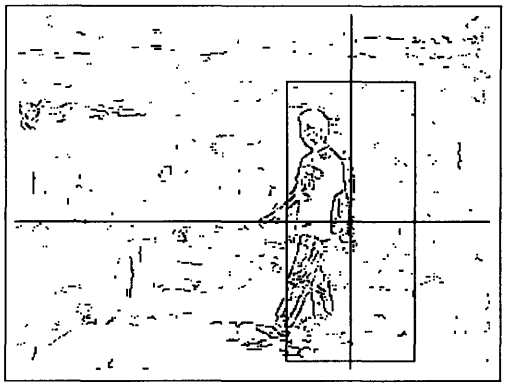


Figure 4-33: Example showing the degradation of the Hausdorff algorithm tracking on the kazu image due to background clutter. The bounding box of the detected person has expanded to be approximately twice the actual width of the person.

4.8 Measuring the Discrimination Ability of the Hausdorff Algorithm

In Section 4.6 questions were raised regarding the ability of the Hausdorff tracker to discriminate between different people. In order to measure the discrimination ability of the Hausdorff tracker, we have constructed an image sequence identical to the kazu sequence except that the twentieth and fortieth image frames have been replaced by the corresponding frames from the magda sequence. Because the sequences were filmed within minutes of each other and from the same camera location, the background is identical in both sequences. Hence the stationary background removal will not be effected by switching between frames from the two sequences.

Figure 4-34 compares the tracking of the Hausdorff algorithm on this artificially constructed kazu-magda sequence with that on the original kazu sequence. It is easy to see that the tracking has barely been effected by swapping these two frames. Looking at the matching process, we see that in several cases a model of one person has been matched to the image of the opposite person with a match score no worse than when matched to the succeeding image of the same person. Figure 4-35 shows the image of the man in frames 39 and 41 and of the woman in frame 40. The image of the man from frame 39 matched the image of the woman in frame 40 with a score of 3.0 pixels, and her image then matched the image of the man in frame 41 with a score of 4.0 pixels. From the histograms of match scores shown in Figure 4-29 we conclude that these are good matches. The two people, however, are very different looking and it is easy to tell from the edge images that they are different. The woman in the magda sequence is much shorter than the man in the kazu sequence and her shirt is highly textured while his is a solid color (so no edges were detected within the interior). Also, they are facing different directions.

This result defeats any hope of using the shape matching inherent in the Hausdorff tracker to solve the person constancy problem for tracking multiple people and distinguishing between them.

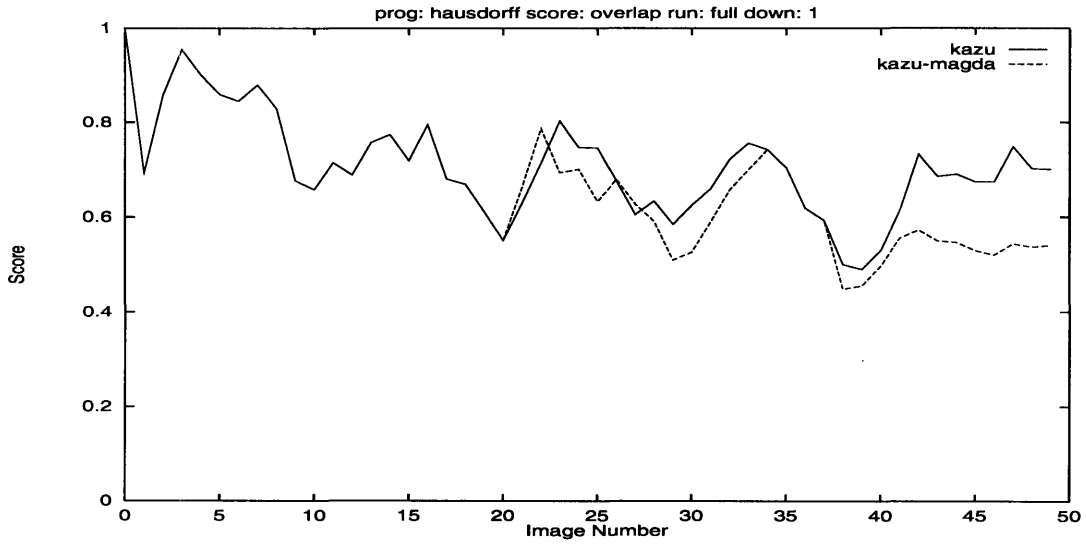


Figure 4-34: Results for the Hausdorff algorithm on the full resolution **kazu** and **kazu-magda** sequences. The results are nearly identical despite the fact that in frames 20 and 40 of the **kazu-magda** sequence, images of a short woman with a highly textured shirt and who is facing a different direction have been substituted. The man from the **kazu** sequence is much taller and is wearing untextured clothing.

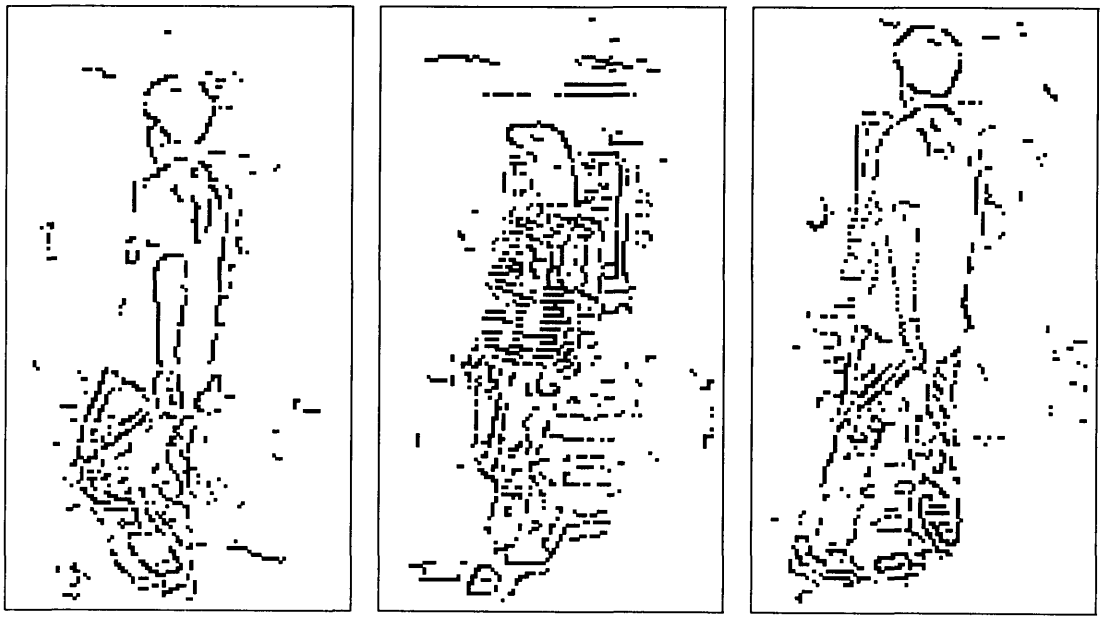


Figure 4-35: Closeup view of the man and woman in frames 39 through 41 of the **kazu-magda** sequence. The left image shows the man in frame 39, the center image the woman substituted in frame 40, and the right image the man in frame 41. The Hausdorff algorithm matched the image of the man in frame 39 to that of the woman in frame 40, and the image of the woman in frame 40 to that of the man in frame 41, giving good match scores.

The examples of tracking multiple people given in [10] were promising but it seems that the algorithm is relying entirely on trajectory information to solve the problem. It does appear that the resolution used in their experiments was roughly 50 percent higher than ours. In their sequences, the image of a person's torso typically occupies nearly the full height of the image and the feet are cut off. At this resolution it would be impossible to have more than two people in view at the same time. We could only afford this resolution and still have a wide enough field of view if we could capture our images at a resolution of 640x480 pixels and process them in real-time, which is unlikely. Simple trajectory information may work if there are only two people, if they only occlude each other briefly, and if the motions are mostly parallel to the image plane, but in the more general case it will likely fail.

4.9 Predicting the New Feature Location in the Correlation Based Feature Tracker

In this section we describe a series of experiments performed to analyze several prediction algorithms to be used in guessing the new position of the feature point for the correlation based feature tracker. We have described in detail in Section 2.1 the motion difference based prediction which we have been using up to this point. The other methods we will be investigating are a number of simple predictive filters. All of these filters are based on extrapolating the new location from a number of past feature locations. First we will describe these methods and then we will analyze how they work.

4.9.1 Simple Predictive Filters

We will be investigating three simple predictive filters. These filters effectively fit a polynomial of degree n to the $n+1$ previously detected feature locations and then extrapolate the new position by evaluating that polynomial at the current time. The filters we will explore are those for values of n of 0, 1, and 2, giving predictions based on a fit to zero-order, linear, and quadratic functions. We will assume that the time between samples is fixed (fixed frame rate). Also, we will consider each of the two image dimensions separately, and can therefore derive the equations for a one-dimensional signal. Let $\hat{x}$ be the predicted position at time t and let x_{t-1} through x_{t-n-1} be the $n+1$ past measurements. For the zero-order and linear predictions, it is easily seen that

$$\hat{x}_{zo} = x_{t-1} \tag{4.7}$$

$$\begin{aligned} \hat{x}_{lin} &= x_{t-1} + (x_{t-1} - x_{t-2}) \\ &= 2x_{t-1} - x_{t-2} \end{aligned} \tag{4.8}$$

Sequence	Frame Rate	Prediction Error (mean/maximum)							
		Zero Order		Linear		Quadratic		Motion Diff	
cereal1	full	2.97	6.08	0.41	1.00	0.79	1.41	3.12	6.40
	half	5.78	12.04	1.53	2.24	0.81	2.00	3.05	5.00
squares1	full	2.94	6.08	0.47	1.41	0.91	2.24	5.21	10.00
	half	5.82	12.04	1.60	2.24	1.15	2.24	4.68	13.00
takeh	full	2.56	11.05	1.25	10.20	1.86	18.11	6.16	16.49
	half	4.64	15.52	3.70	12.65	5.45	24.35	5.86	13.93

Table 4.3: Prediction errors for the correlation based feature tracker using several prediction functions. Both average (mean over the entire image sequence) and maximum errors are given measured as the Euclidean distance in pixels. All experiments were performed at half spatial resolution.

where $\hat{x}_{zo}$ is the zero-order prediction and $\hat{x}_{lin}$ is the linear prediction. Some simple algebra yields the quadratic prediction as

$$\hat{x}_{quad} = 3x_{t-1} - 3x_{t-2} + x_{t-3}. \tag{4.9}$$

The analogous predictions are used for the second spatial dimension. This completes the derivation of our simple predictive filters.

4.9.2 Results

We performed several runs of the correlation based tracker in which we compared the various predictions in each step with the true feature displacement. For each prediction function, we computed the maximum Euclidean error over the sequence as well as the mean error. The results are summarized in Table 4.3 for three sequences run at half resolution and at full and half frame rate, for each of the four prediction algorithms.

Overall, the simple linear extrapolation appears to perform the best and the motion difference prediction does the worst. The difference is quite significant, the error made by the linear prediction is two to three times less than that made by the motion difference on average. Surprisingly, the quadratic extrapolation only does better than the linear prediction on a few of the controlled motion cases. The object in these sequences is undergoing constant acceleration and we would expect the motion to be best fit by a second order polynomial. It appears, however, that the imperfections in the controlled motions introduced enough noise for a linear model to fit more closely. The simple extrapolating predictions are not as accurate on the takeh sequence as on the controlled motion sequences, however, they are still very accurate.

Experiments were also performed on Kalman filters (see [7] and [6]) using constant velocity and constant acceleration models. We found, however, that these more general filters performed best when the parameters were set to yield exactly the simple linear extrapolation filter described above.

Given these new results, and the problems with the motion difference prediction discussed in

Sections 4.1 and 4.6, we conclude that using a simple linear extrapolation based on the previous two detected feature locations would be much wiser and give better predictions. The motion difference prediction is highly dependent on the texture of the object and the speed of its motion, as we have seen. Recall the downward bias exhibited on the `squares1` sequence due to the lack of texture on the object and the absence of an intensity difference between the red and green squares. There is also a significant bias when the object position overlaps from frame to frame. If the object is moving slowly, there is a significant bias toward the leading edge of the object. Consider a textured square moving slowly down and to the right. Figure 4-36 shows possible difference images D_{t-1} and D_t at the previous and current times, and the motion difference MD_t at the current time. Clearly the center of mass of the detected motion difference pixels is biased downward and to the right of the square's center. In order to avoid these types of biases, the object must be moving fast enough not to overlap in three consecutive images. As we mentioned previously, this is a difficult condition to satisfy, especially when there are multiple moving objects. Finally, if the object is moving with this speed, the prediction windows must at the very minimum be at least three times as wide as the object. For accuracy when tracking multiple people, this requires there to be clear space between each two people of at least the width of two people. Since these conditions are very limiting, we find that a simple linear filter as described above should be preferred to the motion difference prediction.

4.10 Lighting Changes

This section describes a number of experiments we have done to determine the effect of lighting changes on the three algorithms. So that the lighting changes were precisely controlled and to make things easiest, we decided to artificially simulate lighting changes by linearly scaling the image intensities. The experiments involved decreasing the image intensities by a factor of fifty percent, over the course of some number of image frames. The durations over which these changes were programmed to take place were chosen to be 1, 2, 3, 5, and 10 frames. The results of these experiments for each algorithm are described below.

4.10.1 Hausdorff Algorithm

We observed no significant effect due to the lighting changes in our experiments on the Hausdorff algorithm. Since the algorithm is edge-based and the fifty percent reduction did not noticeably effect the edges detected, we expect little change other than the introduction of a small amount of noise on the tracking. If we had used edge detection thresholds scaled relative to some measure derived from the image, the effect could have been even less. The results for the `takeh` sequence are shown in Figure 4-37. These results are typical for our sequences, except that the performance actually increased for the `kazu` and `magda` sequences. The tracking expanded less into the background

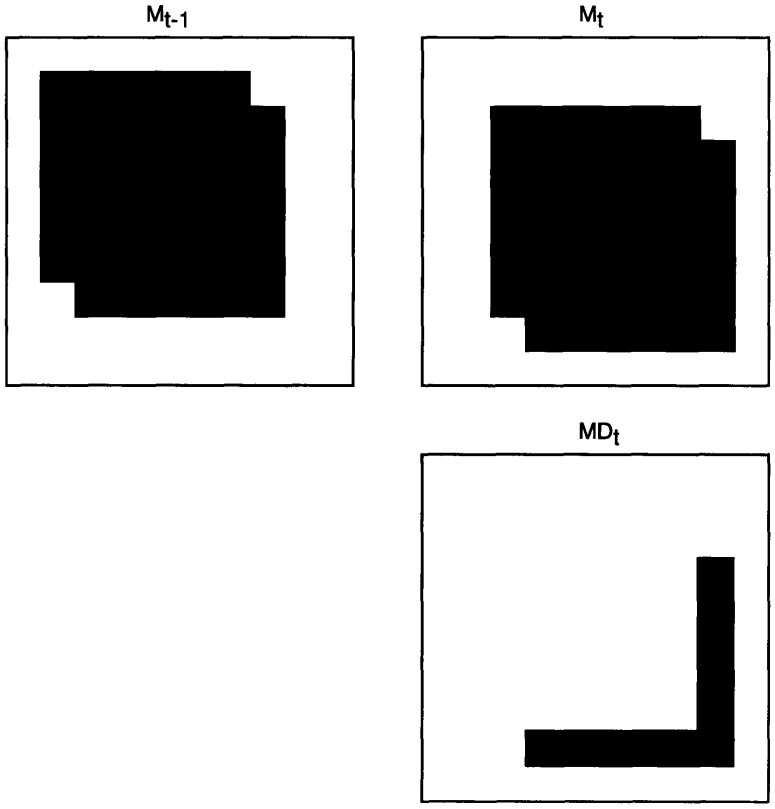


Figure 4-36: Example of bias in the motion difference prediction employed by the correlation feature tracker. Top row show previous and current difference images for a square object moving down and to the left. Bottom row shows the computed motion difference. Clearly the center of mass of those pixels is not the center of the square in the current image; there is a significant bias toward the bottom right.

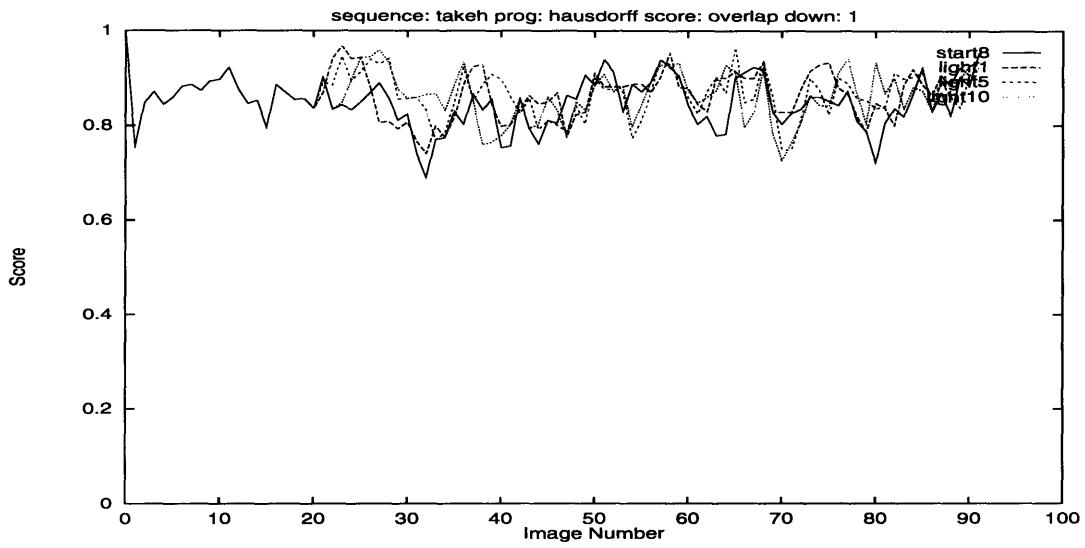


Figure 4-37: Results of the lighting experiments for the Hausdorff algorithm on the full resolution takeh sequence. The results for no lighting change are compared with the results of a fifty percent decrease in intensity programmed over the course of 1, 5, and 10 images, all starting in the twentieth image frame.

because fewer background pixels were detected after the lighting decrease. Recall that in Section 4.7 we observed problems due to high background texture in these sequences. This suggests that we should increase the edge detection thresholds when using this algorithm in settings with highly textured backgrounds.

Because the lighting was decreased, it is only possible for edge pixels which would have been detected to disappear. If new edge pixels were to appear, the stationary background removal might fail to remove newly appearing background pixels and this may cause problems. In addition, detecting more background edges even after the change could make it more likely for the tracking to expand into the background. We ran an additional experiment in which the image intensities were increased twenty five percent over the course of a single frame. We did notice a small degradation in the tracking, however the effect was not immediate but took place over the remainder of the sequence, indicating the later explanation above. Figure 4-38 shows the results of the lighting increase on the takeh sequence.

4.10.2 Motion Based Algorithm

There were two effects observed due to lighting changes to the motion based algorithm. The first has to do with effects on the difference image and optical flow steps during the lighting change. This effect is exemplified by the performance on the cereal1 sequence. Figure 4-39 shows the performance for a fifty percent decrease in image intensity over the course of 1, 2, 3, and 5 image frames compared with the performance for no lighting change (labeled start8 — note that the lighting runs all started

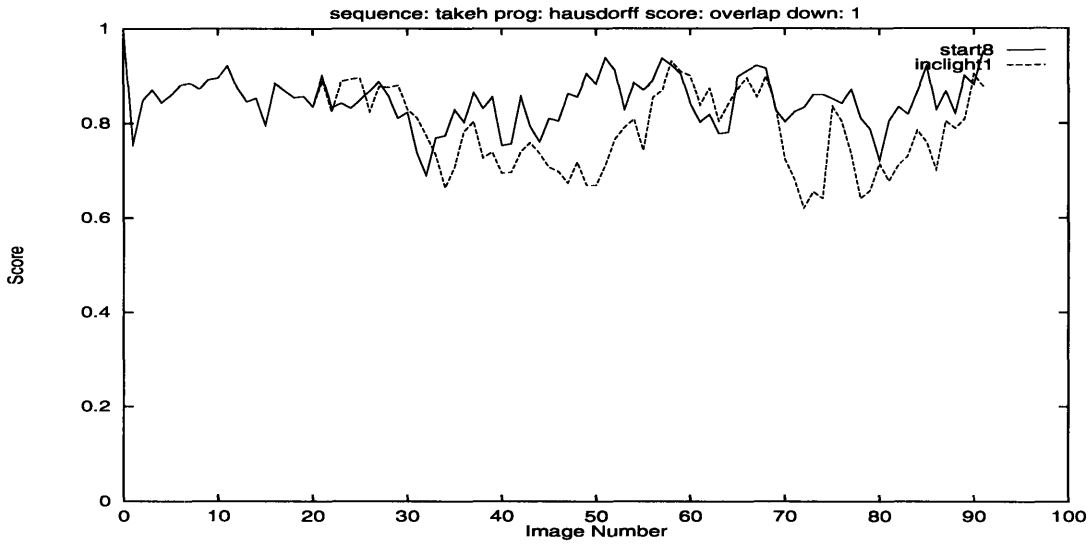


Figure 4-38: Results of the experiments involving a lighting increase for the Hausdorff algorithm on the full resolution `takeh` sequence. The results for no lighting change are compared with the results of a twenty five percent increase in intensity between the twentieth and twenty first frames.

with the eighth image frame in each sequence for reasons described in Section 4.2). Concentrating on the drops in performance starting in frame 10 where the lighting change was initiated, we see that the effect is worst for the fifty percent change over two and three frames. This change over 1 frame has had about half the effect as over two frames, and the effect for that change over five frames is negligible compared to the control. Note that the effects seen at frame 25 are due to the object stopping and were discussed in Section 4.1.

When the lighting changes quickly, we expect the difference image to detect change across the entire image including the background. The hope is, however, that the optical flow will determine which image locations were actually in motion and which changed due to lighting or shadows. The difference is meant only to speed the optical flow computation by ignoring large areas where there was no change. Figure 4-40 shows the difference images and motion detected in frame eleven of the lighting runs (the first frame of each run where the lighting was effected) in which the light decreased by fifty percent over 1, 3, and 5 frames. These runs will be called `light1`, `light3`, and `light5`, respectively.

The difference image for the `light1` run comprises nearly the whole image, as expected. Nearly all the pixels changed by more than the 20 grey level difference threshold during a fifty percent change. As the period over which the lighting change takes place increases, however, fewer and fewer pixels are detected as different. With the change over three frames (16.7 percent change per frame) much less background is detected as different and by the time the change is spread over five frames, change is detected at only a handful of pixels. Because the threshold is a fixed number of grey values, rather than a proportion change, it is more likely that change will be detected in bright

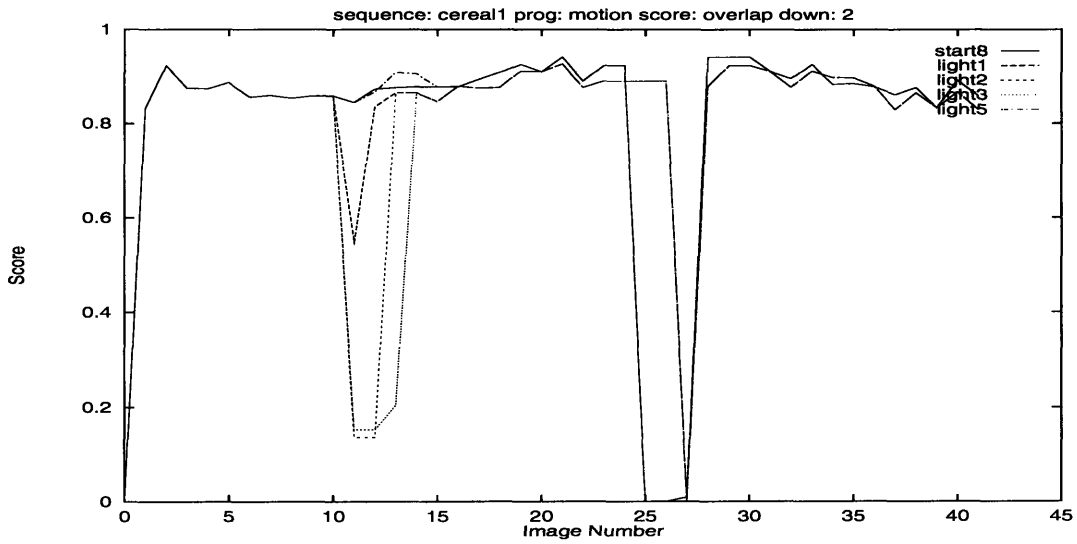


Figure 4-39: Results of the lighting experiments for the motion based algorithm on the half resolution `cereal1` sequence. The results for no lighting change are compared with the results of a fifty percent decrease in intensity programmed over the course of 1, 2, 3, and 5 images, all starting in frame ten.

image locations than dark regions. This could be considered a problem — if so however, the fix is easily made by making the threshold relative to the pixel intensity. Finally, with a threshold of 20 grey levels, we expect no background change for a fifty percent lighting change over more than 6.4 frames by solving

$$255 \text{ levels} \cdot \frac{50\%}{N \text{ frames}} < 20 \text{ levels}.$$

The data shown support this claim.

Looking at the motion detected, we see that for the rapid lighting change in the `light1` run, motion is for the most part only detected on the cereal box which is moving in the center of the image, as we expect. However, with the change spread over three frames, motion is detected almost everywhere a difference was detected, including the background. Because the background in this sequence is flat (untextured) and there is an intensity gradient, for a slower change we have again encountered the false motion problem discussed in Sections 4.3 and 4.6. A stationary optical flow patch from before the change is darker after the change, but a neighbor to the right which had been brighter is now close enough in intensity that it is now a good enough match, and motion is detected. This problem is not exhibited when the change is fast enough because the distance that would be required for a match is far greater than the flow search radius. It is also not exhibited when the lighting change is slow enough because no difference is detected and therefore flow is not computed. Hence, false motion detection causes the effect of lighting change to be worst for intermediate changes. It would probably not be hard to detect the condition where a difference is detected over the whole image and wait to resume tracking when the lighting stabilizes. The problem is only serious in the case of a drastic change in lighting levels.

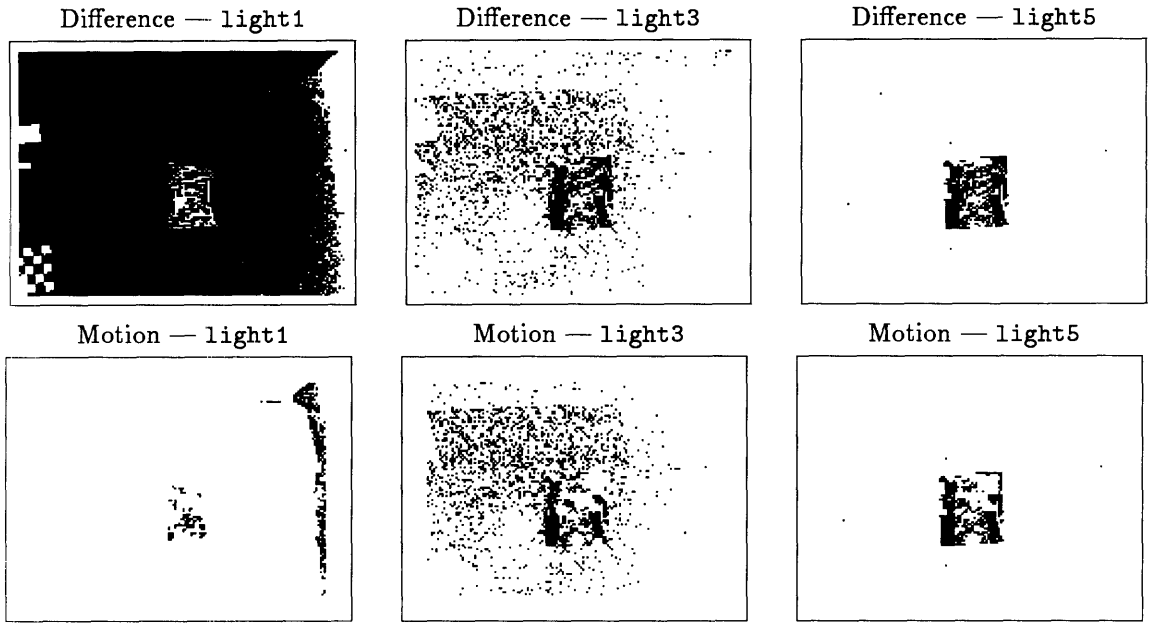


Figure 4-40: Difference images and motion detected for the motion based algorithm run at half resolution. These images are for frame 11 of the lighting runs `light1`, `light3`, and `light5`, in which the intensity was decreased by fifty percent over the course of 1, 3, and 5 frames, respectively. Frame 11 is the first frame effected by the lighting change.

The second effect we have observed due to lighting changes on this algorithm is much worse fragmentation of the detected motion into many connected components at low lighting levels. This effect is shown in Figure 4-41. After the decrease in image intensities by fifty percent, the tracking relative to the control run is significantly worse. Due to an absolute rather than relative difference threshold, it takes greater relative change for a darker pixel to be detected as undergoing a 'significant' difference than a bright pixel. When the intensities are decreased, therefore, it causes fewer motion pixels to be detected, especially on the darker regions of the object. Hence, the motion pixels more easily fragment into multiple connected components.

We have seen two effects on the motion based algorithm due to lighting change, one due to false flow detection, and the second due to higher sensitivity to change in bright pixels. The first again casts doubt on the efficacy of the optical flow computation. In Section 4.11 we will explore this further. The second problem should be ameliorated by using a relative threshold for the difference detection.

4.10.3 Correlation Feature Tracking Algorithm

There were also two major effects to the correlation based feature tracker due to lighting changes. The first pertains to the motion difference based prediction algorithm. As we saw for the motion based algorithm, during rapid enough lighting changes, nearly the whole image is detected in the

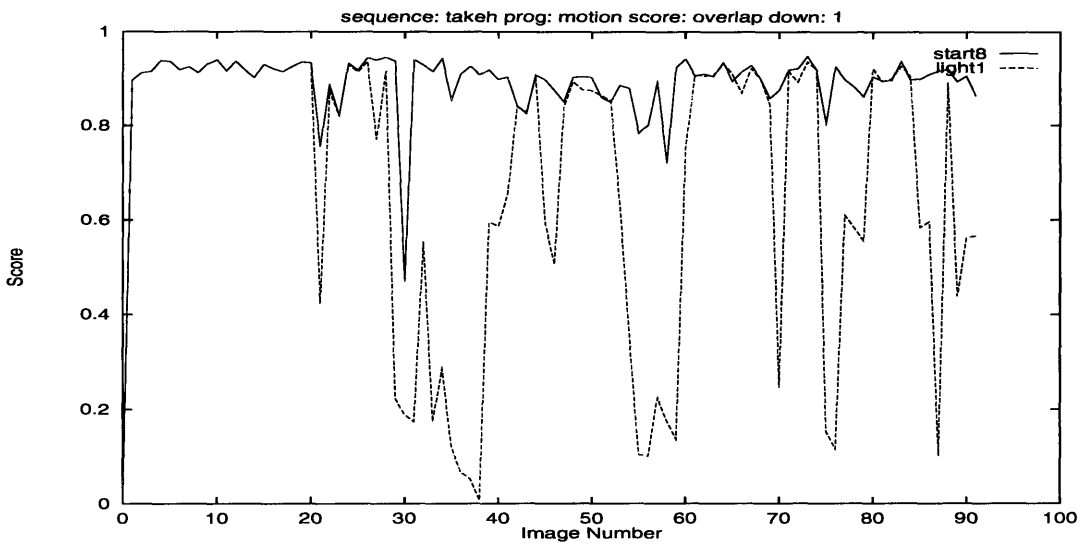


Figure 4-41: Results of the lighting experiments for the motion based algorithm on the full resolution takeh sequence. The results for no lighting change are compared with the results of a fifty percent decrease in intensity between frames twenty and twenty one.

difference image. This causes nearly the whole image to appear in the motion difference, and hence the predictions made for the feature's new location are invariably at a displacement of very nearly $(0, 0)$ from the previous location during fast lighting changes. Since the difference threshold is a fixed constant number of grey levels (as in the motion based tracker) rather than relative to intensity we are more likely to detect differences in brighter regions as the duration of the lighting change increases. Hence, for medium duration lighting changes, the prediction is biased toward the bright image regions near the feature's detected location. A strong bias in the prediction could cause the tracker to lose the feature because of the limited search area about the prediction. We did not notice any cases where this was actually the cause of tracking failure, however. For gradual changes, the difference image will not detect any change. With a threshold of 15 grey levels, a fifty percent reduction in intensity over more than 8.5 frames would not be detected.

The second effect is that with a fast enough lighting change, the feature template is likely to jump to some new feature patch which previous to the lighting change had been much brighter, especially if there was not a great deal of strong texture on the feature patch. For example, on the takeh sequence, the tracker switched from tracking the man's dark pants to tracking his shirt after the fifty percent reduction in intensities in a single frame. Prior to the lighting change, the man's pants had an intensity of approximately 55 and his shirt had an intensity of 110. After the change, the shirt's intensity became almost identically that of the pants before the lighting change. For less rapid lighting changes, this problem was greatly reduced. Computing a normalized cross-correlation and/or matching on pre-processed images [4] could greatly reduce this problem.

Other than these two problems, the tracking was very good for this algorithm through our lighting

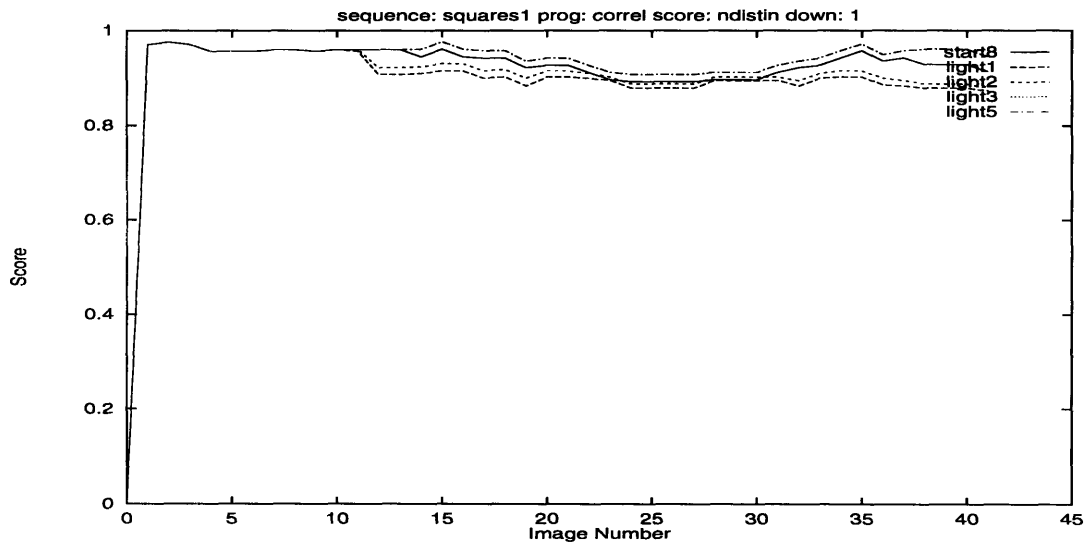


Figure 4-42: Results of the lighting experiments for the feature tracking algorithm on the full resolution `squares1` sequence. The results for no lighting change are compared with the results of a fifty percent decrease in intensity of durations 1, 2, 3, and 5 frames, starting in frame 20.

changes. For example, on the `cereal1` sequence, the tracking for all of our lighting experiments at full resolution was identical to that with no lighting change. At lower resolutions, the tracking was different only for the most rapid lighting change, in which case the tracking jumps onto the background which had previously been twice the intensity of the cereal box. The reduced texture on the box at lower resolution caused this. Tracking on the `squares1` sequence was nearly identical to the control at all resolutions. Figure 4-42 shows the tracking for this sequence at full resolution.

In summary, only the most drastic lighting changes had any effect on the correlation based feature tracker.

4.11 Optical Flow Computation in the Motion Based Tracker

Our final set of experiments was performed to determine if the motion based algorithm gains anything by the optical flow computation it performs. Recall that in Sections 4.3 and 4.6 we found motion to be detected where in fact there was none. When the object was moving fast enough to exceed the optical flow search radius, the algorithm detected the background at the new object location to have expanded out of the way of the object. Thus in some cases the computed flow is inaccurate. Further, in watching several runs of the algorithm, the motion pixels detected appeared to be nearly identical to the detected difference pixels. The only reason for actually computing the flow seems to be for lighting invariance, but as we saw in Section 4.10, this does not work terribly well. Here, flat background regions were detected to have moved to locations that had been brighter. Hence, we wonder if the algorithm might work just as well using the difference image directly as the detected

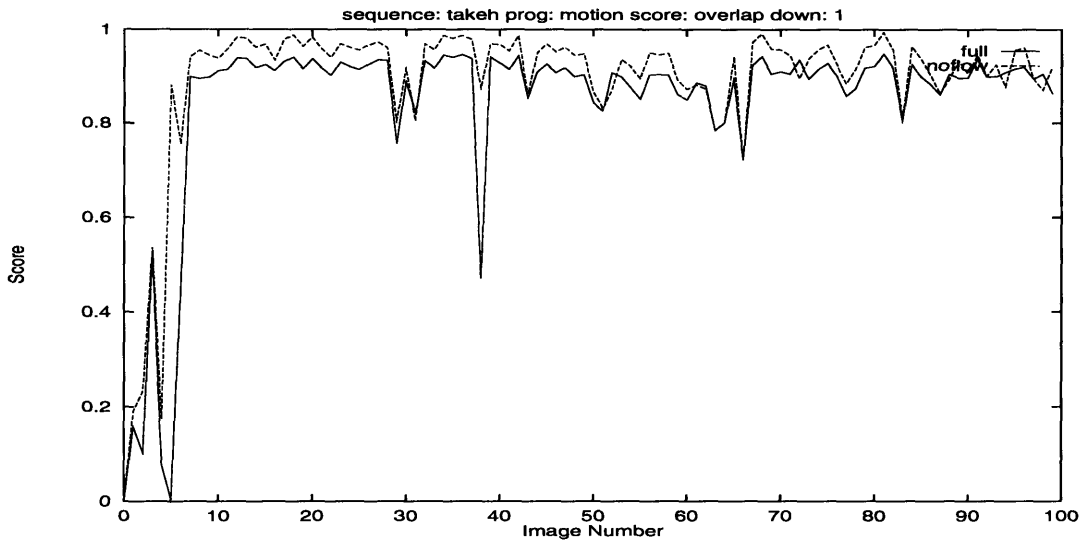


Figure 4-43: Results of the optical flow experiments for the motion based algorithm on the full resolution **takeh**. The results are better with the optical flow computation turned off.

motion. This would allow us to either run the algorithm at a higher frame rate or at a higher resolution, or some combination of both. We have already seen that the tracking is greatly improved by increased resolution. The results on higher frame rate were less conclusive, however, with extra time some other algorithm could operate in parallel.

To determine whether the flow computation is in fact useful, we ran the motion based algorithm on our sequences at all three resolutions, with and without the lighting changes, with the optical flow turned off and compared the results to our earlier experimental runs. In all of these comparisons, we did not detect one significant degradation due to not using the optical flow. In fact, in some situations the results were actually better. The results were a little worse on the most rapid lighting changes (fifty percent reduction over one frame), however, we have already discussed the fact that this condition should be easy to detect and therefore does not present a problem.

Figure 4-43 shows the results of the full resolution runs of the motion based algorithm with and without the optical flow. The results are in fact better for the run which did not use any optical flow. Figure 4-44 shows results with and without optical flow for a rapid decrease in lighting. The image intensities are reduced by fifty percent between frames ten and eleven. The results are only slightly worse for the run without optical flow.

Hence, we conclude that the optical flow computation is unnecessary in the motion based algorithm. Given the timing data listed in Section 4.12, we see that this provides a huge computational savings at high resolutions. The optical flow computation is the most computationally expensive aspect of the motion based algorithm, having complexity $\Theta(N_x N_y R_S^2 R_P^2)$ where N_x and N_y are the image dimensions, and R_S and R_P are the search radius and patch radius, respectively. Since all of these factors are linearly proportional, we see that the algorithm has complexity $\Theta(N^6)$ where N is

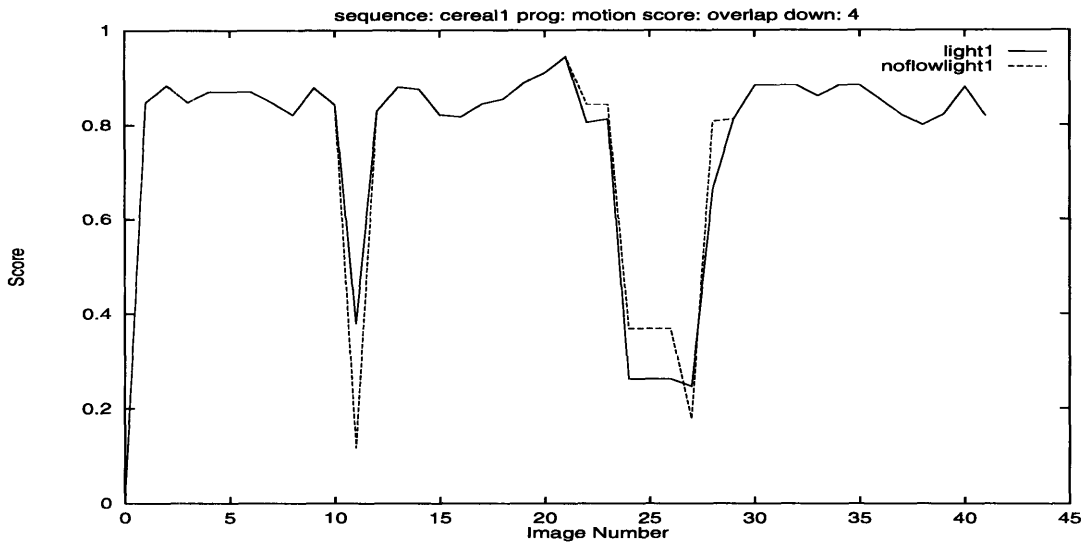


Figure 4-44: Results of the optical flow experiments for the motion based algorithm on the quarter resolution `cereal1` sequence, with an artificial lighting decrease of fifty percent between frames ten and eleven. The results are only slightly worse with the optical flow turned off.

one image dimension. Eliminating this step helps considerably.

4.12 Timing

Timing data has been collected for all of the experimental runs. Designing a person tracking algorithm which runs in real-time is a requirement for the applications we envision and hence must be one of our considerations. Table 4.4 summarizes some timing data for the algorithms we have tested on three of our sequences and at several resolutions. The column labeled 'Motion/No Flow' refers to the experiments done in Section 4.11 in which the optical flow computation was turned off in the motion based algorithm.

Assuming we want the tracking system to operate at a frame rate of at least 5 to 10 frames per second, and assuming these algorithms could be sped up by a factor of ten to twenty by a combination of coding them more efficiently and implementing them on embedded DSP processors[1] we can conclude that any algorithm and image resolution listed in Table 4.4 slower than a few seconds per frame could not feasibly be implemented to run in real time. This eliminates consideration of the motion based algorithm and the correlation based feature trackers at full resolution. All other combinations are sufficiently fast that they could be run in real-time.

This ends the presentation and analysis of our experiments on the three tracking algorithms. In the next chapter, we summarize the important results and their implications for the use of each of

[1] Our research group is currently using several embedded computers based on the Texas Instruments TMS320C31 processor designed and built by Chris Barnhart

Sequence	Resolution	Algorithm			
		Hausdorff	Correlation	Motion	Motion/No Flow
`can4`	full	437	17218	6705	303
	half	—	1179	203	39
	quarter	—	87	19	9
`cereal1`	full	564	18946	14600	543
	half	—	1264	406	55
	quarter	—	94	28	13
`takeh`	full	954	4896	19077	651
	half	217	354	527	68
	quarter	—	35	36	14

Table 4.4: Timing data for the algorithms on several of the `start8` runs (see Section 4.2 for details). Times are given in milliseconds per frame and were measured on a Sun SPARCStation 5. Values are not specified for cases where the tracking was not satisfactory. The times listed for the Hausdorff algorithm do not include edge detection.

these algorithms in a multiple person tracking system. We then propose future experiments and some ideas on an algorithm to solve the person tracking problem.

Chapter 5

Conclusions and Future Work

We begin by reviewing the key results for each algorithm obtained from the experiments described in the last chapter. Then, conclusions are drawn about the prospects for building a person tracking system based on these algorithms, in light of our results. Finally, a framework for an algorithm is proposed and further experiments to address the remaining open problems are discussed.

5.1 Review

5.1.1 Correlation Based Feature Tracking Algorithm

We saw that the correlation based feature tracker worked well at all of the resolutions tested. The performance definitely improved significantly as the resolution was increased. However, the computation required at the highest resolution tested was well beyond what we can afford for a real-time implementation. One good feature of this algorithm is that it was unaffected by a lack of motion, unlike the other two algorithms. As we have noted, we expect a full range of human motion speeds in our application, from sitting still to walking quickly and making rapid hand gestures.

A major drawback of this algorithm is the slow drift problem. We saw in several experiments that the algorithm tends to slowly drift off of the feature which it began tracking, due to sub-pixel motion and any shape change resulting in non translation-only motion of the feature patch. It would probably be possible to use a coarse-to-fine strategy to gain the advantages of operating at higher resolution, namely less drift and better support for matching, while still running in real-time. Also, interpolation in the SSD space should give some additional sub-pixel resolution. However, we would still expect drift problems over long time periods, especially due to non-rigid motion.

Because of this unavoidable drift, we foresee that over long time periods the algorithm will eventually lose track of the original feature. There is no mechanism to draw the tracking back toward the original feature and in this sense the algorithm is somewhat unstable. Recall that small

amounts of noise introduced due to the lighting changes sometimes caused the tracking to diverge. Also, there is no mechanism to detect whether the algorithm is still tracking the original feature after several frames. If the feature drifts too far, it is likely that the feature patch will contain some background. Eventually, chances are good that the algorithm will begin tracking the background, especially if the background is highly textured.

The limited search area requires the predictions to be relatively accurate. Computationally, it would be prohibitively expensive to use a very large search area. It seems that running the algorithm at a high frame rate would be beneficial to limit both the required search area as well as the amount of shape change possible between frames. From our data on the speeds of human motions we can determine a reasonable bound on the search radius given the image resolution and frame rate. It is likely that the feature may still move outside the search radius on occasion, for example if an image frame is missed. Detecting and recovering from this case should be addressed. A match threshold would likely help significantly to detect this problem.

Our experiments on predicting the new feature location show that a simple linear extrapolation from the previous two detected locations performed best. This model assumes a constant velocity motion. The motion difference prediction was found to have many problems with untextured or slow moving objects. If objects are moving fast, however, the windows within which the center of mass of the motion difference pixels are computed to form the prediction must be very large. This will cause problems for tracking multiple people because it would severely limit how close people can approach each other.

Size and placement of the feature window is another difficult problem with this algorithm. A feature patch selected must have significant texture so that a single clear minima is found in the SSD space as it is tracked. The patch should be placed on a relatively rigid feature and should not contain occlusion boundaries. As we saw in Section 4.7, placing the feature patch at the waist of a person walking sideways gave very poor performance due to the occlusion by the hand and to the non-rigidity of the leg motion. This greatly accelerated the drift problems with this tracker. When placed on the person's head, the tracking was much better. Increasing the template size gives a larger support and improves the tracking. A very small template will likely match nearly everywhere. However, larger templates are more susceptible to drift due to non-rigid motion. They are also more likely to straddle an occlusion boundary.

Finally, the algorithm was found to be quite robust to lighting changes, except for the motion difference based prediction. Only for very rapid lighting changes did the feature tracker lose track of the original feature and begin tracking something else.

5.1.2 Motion Based Algorithm

The motion based algorithm gave a good segmentation of the moving object provided the whole object was moving but was not moving too fast. The performance was best at higher image resolutions, however even at quarter resolution, the tracking was still often sufficiently good. At full resolution, the tracking was too slow to be useful in real-time. If the optical flow computation is eliminated, the times required at full resolution are no longer a problem. Given our finding that the algorithm does not gain anything from the optical flow, we suggest that it should not be computed.

The algorithm fails when there is no motion, very slow motion, or when only parts of the person being tracked are moving, for example when making hand gestures. The expected problems in the optical flow step due to the motion being too fast were much reduced by false motion of the background being detected. With the optical flow eliminated, there is likely to be slightly more smearing of the person's detected position to be the union of her positions in the previous and current frames. This can be reduced by using a sufficiently high frame rate. However, if the frame rate is too high, the algorithm will fail because there will barely any motion between frames. The ideal frame rate really depends on the speed at which the people being tracked are moving.

Several conditions all cause the algorithm to break in much the same way. Lack of texture on the object, object color similar to the background, low light levels and darker regions, low resolution, slow speed, and body parts moving at different speeds all cause sparse motion detection and result in the grouping algorithm fragmenting the motion of a single object into many components. This causes only a part of the object to be detected. Without texture, motion is primarily detected at the object boundary and not in the interior. In regions of the object which are nearly the same color as the background, no motion is detected. These problems, as well as the problems with different motion speeds are not easily solved. The others are not as difficult. At low light levels, change is less likely to be detected in darker regions. This problem can likely be reduced by using a difference threshold which is relative to the pixel intensity. Of course, for very dark regions we must be careful that the threshold does not go so low that change is always detected. Not computing optical flow should leave plenty of time to run the algorithm at a higher resolution.

Finally, we have seen that the algorithm's performance during lighting changes is worst for medium duration changes. Without the optical flow, however, we expect it to be just as bad for rapid changes. During a lighting change, motion is detected everywhere, making it impossible to segment the objects based on motion. It should be possible to detect this condition, however, and inhibit the tracking until the lighting stabilizes. This could cause a problem if the lighting change lasts too long. The algorithm is not really solving the person constancy problem well at all, basing decisions on which motion regions correspond over time on which region in the previous frame was closest spatially to the current region. Thus, if the people have moved considerably while the lighting was unstable, this will fail.

5.1.3 Hausdorff Algorithm

Although we expected the Hausdorff algorithm to be the clear winner among the three tracking systems we have studied, this was not actually the case. In practice it was never significantly better than the other two, and we have found many problems in several hard cases. Under the right conditions, however, the tracking seen by the Hausdorff algorithm was quite good. It has the advantage of searching over all possible translations for a match with the object model in each frame. Further, being edge based, it is very insensitive to lighting changes; a fifty percent change in image pixel intensities had essentially no effect.

It is not clear, however, that searching everywhere is a good thing. For example, in some of our experiments, we observed that as the object stopped moving, the object model would be matched to some remaining background fragments not fully removed by the stationary background removal. Often these matches were very far from the previous object location, on the opposite side of the image. Knowing how fast humans can move and the sampling rate at which we are operating are strong constraints which can be used to limit the search and avoid false positives. Since we presumably need to sample quickly enough to avoid very large shape changes in the people we are tracking, it seems reasonable to also use the fact that they can not have moved far as well.

Similarly, being edge-based, while good for lighting invariance, does have the disadvantage that most of the original image information has been thrown away in the edge detection process. This is evidenced by the significantly higher resolutions required by the Hausdorff algorithm to track well. We have found that object sizes of at least 50x100 pixels are needed to achieve good tracking, probably much more for non-rigid objects or objects with little texture and hence fewer edge pixels. The other algorithms worked even for the smallest resolutions tested, albeit with lower performance.

One of the major problems we observed for this algorithm was that with very complex and cluttered backgrounds the tracking degrades significantly when the object model expands to include background edge fragments which were not fully removed. We have seen the object model grow to three times the size of the person in fewer than 50 image frames. Another problem comes from a lack of motion or very slow motion, especially when the object slows to a stop gradually. Over time, the object models become more sparse as the object comes to a stop. When the object actually stops, even briefly, and is removed by the stationary background removal, the previous very sparse model will match anywhere, usually to some background fragments. These fragments are then tracked. We attempted to adjust the algorithm's model density parameter but were unable to find a good value which both reduced model expansion into background and allowed brief stoppage of motion.

Finally, we had hoped that the shape matching done by this tracker would be able to distinguish between different people and combined with the model library of canonical views would help us to solve for the correspondences between several tracked people through time. We have found, however, that the shape matching can not discriminate between even very different looking people.

It appears, looking at the matching process, that the algorithm is only looking for a cloud of points of very roughly the right size and density. At much higher resolutions, there may yet be some hope, however, we are not sure we could run the algorithm in real-time at such sizes, especially once the time for edge detection is included.

5.2 Conclusions

Our experimental results have several implications for building a system to track multiple people. First and foremost, all three algorithms have serious problems. Any one taken alone will certainly not solve the problem given the various ways we have discovered in which each breaks.

Also, we have not come at all close to solving the problem of person constancy. The motion based algorithm's only notion of a tracked object being the same person between image frames is by finding the detected motion region in the current image closest to the location the object was detected in the previous image. At a reasonable frame rate this will work for one person but problems arise as soon as there are multiple people. For example, if one person stops moving, the motion of the second will be the closest moving thing, and would be tracked as if it were the first person. A better notion of object 'sameness' over time is needed. The feature tracking algorithm probably exhibits the best approximation to this, at least from frame to frame, because it matches the appearance of an image patch between frames. However, the problem really needs to be addressed over longer time scales. We have seen that the Hausdorff algorithm does not solve this problem either.

Introducing people passing nearby or behind one another further complicates the situation. In either of these situations, their detected motion regions in the motion based algorithm will merge into a single region. When the motion regions later split apart, the algorithm would need to again begin tracking each of the people and determine which is which. Currently, there is no provision for this in the algorithm. With the correlation based feature tracker, passing nearby is not much of a problem, but occlusion causes any patch on the occluded person to disappear. Somehow the patch must then be re-acquired when it re-appears, despite the fact that its appearance might be much different. Detecting the occlusion and re-acquiring the feature are not currently handled by the algorithm. These situations clearly put a heavier burden on the person constancy computation, requiring it to function over longer time scales and be insensitive to shape change. They also introduce problems of their own.

Looking at the problems with the Hausdorff algorithm, we do not think it would be a good framework on which to base our multiple person tracking system. The need for high image resolution means we would at best be operating near the bare minimum object sizes required for effective tracking. The problems with lack of motion and tracking background do not seem easily fixed, and the algorithm does not capture object shape well enough to distinguish between objects. There are

also questions about the stability of the algorithm. Finally, the system is much more complicated than the other algorithms, making it harder to understand and reason about, as well as to port to other hardware environments, especially an embedded platform. There does not seem to be any evidence that the extra baggage is worth it for our application.

Turning to the other two algorithms, however, we see that many of their strengths and weaknesses are quite complementary. The correlation based feature tracking algorithm does not track the whole object, but rather a small feature, while the motion based algorithm gives a relatively good segmentation of the whole object. The feature tracking algorithm is able to deal with a lack of motion and give good estimates for the inter-frame displacements of points on the object, while the motion based algorithm has much trouble with very slow motion. When there is motion, the motion based algorithm is very stable in the sense that if it is perturbed slightly it quickly converges back to the true segmentation. On the other hand, the feature tracker will always exhibit a slow drift away from the original feature and the effects of small perturbations are not reduced over time. Finally, while the motion based algorithm has no notion of object constancy, the feature tracker with a large enough support region does give some amount of frame to frame appearance matching. These complementary aspects of the algorithms suggest that we might combine them to gain the benefits of both. In the next section, we explore this idea further and discuss the hurdles that would still remain.

5.3 Future Work

Here we suggest several ways that the motion based segmentation and correlation based feature tracking algorithms could be combined to take advantage of the strengths of both algorithms. We envision running both algorithms in parallel, each using information computed by the other to guide various aspects of its own computation. Basically, we can use the motion based algorithm to give us the segmentation of the people, at least while they are in motion, and at the same time we use the feature tracker to track some number of features on each person. These features give some notion of inter-frame person constancy and we can use these features for the correspondences in simple cases. The feature tracking also gives better estimates of the people's velocities given their larger support. These estimates can be used to detect and continue to track people who stop moving, even though the motion segmentation will be giving little to no information. The velocity estimates might also be used to adjust the frame rate of the motion segmentation. Finally, the drift problem for the feature tracker could be solved by guiding the placement of the feature patch back onto the person during times when the motion segmentation is good, for example, when the people are moving fast enough. Using the assumption that the person is standing upright, we might even be able to reliably position the feature on the person's head or some other robust feature location.

Further research to study the viability of these ideas is clearly necessary. Better information on the optimal time scale to use with the motion based algorithm is needed. Using only the difference image as detected motion should be fast enough to run the algorithm quickly at high resolution.

For the feature tracker, we must first introduce a mechanism for the tracker to detect cases where no good match is found for the feature, instead of choosing a best match whose appearance does not at all resemble the feature. Cases where this is necessary include occlusion, significant shape change, and a limited search region. Using a match threshold on the minimum SSD value found should help greatly here. Reasoning about the shape of the match SSD surface may also be useful to detect cases where there is not a single clear minima in the SSD space (see [1]). Incorporating the techniques described in [21] could also give some notion of whether the same feature is being tracked over longer periods of time.

Next, the tradeoff between patch resolution and frame rate needs to be better understood and we need to investigate the speed and accuracy advantages of a coarse-to-fine strategy and interpolating the match location to sub-pixel accuracy. The number of feature patches to use per person and where to place them also needs to be addressed. Also, we need to determine if the features can be guided using the motion detection to keep them tracking the people for long time periods. It would also be interesting to see if the feature trackers might be able to give some evidence for which person passes in front in situations where occlusion takes place. The features for this person should continue to be tracked well, while those on the occluded person should not be.

Given that these ideas still look promising after further research, we have still not solved all of our problems. First, the motion region fragmentation problems have not been addressed for the motion based algorithm. Using constraints on the size of people in the room and smoothing the bounding box sizes over time should help group together the motion regions from a single person that have become separated due any of the reasons seen in Chapter 4. If we allow only gradual size change and extrapolate each previous bounding box position and size to the current frame, we should be able to reliably group the fragmented connected motion components in cases where there is no occlusion. Further, the bounding box size should be allowed to change at much slower rates when the person's motion is detected to be slow. Currently when a person slows to a stop, the box shrinks to a single pixel before there is actually no motion, losing the previous information about the person's size. Finally, other room constraints, such as the positions of the entrances, floor, walls, and other immovable objects could be incorporated as well to aid in grouping the detected motion into people.

Better notions of person constancy over longer time periods also need to be investigated. We believe that color histograms[23] of the image pixels detected on each person might be a relatively simple and fast way to solve the problem in many situations. The pixels to histogram could be selected by the detected motion and the histogram updated with each frame, except care should be

taken in cases of occlusion. This solution does rely on the people in the room wearing distinctive clothing; for example, it would not work in a military setting where all participants are in uniform. Color histograms should be very fast to compute and match even at the high image resolutions which will be needed in order to have enough pixels from the image of a person to histogram.

Finally, reasoning about occlusion and people being nearby one another is needed to detect when these conditions occur and to resume tracking both people when the conditions no longer exist. Using the velocities of the detected people should allow us to predict when occlusion or merging of the components from multiple people due to proximity in the image plane will occur. At this point, updating of color histograms for the people can be inhibited and we can carefully try to determine which person is in front using the feature trackers, color histograms, and sizes of the people (the farther one is likely to appear smaller in the image plane). When the components later separate, we can determine which is which, again using the color histograms and the fact that the one in front should have been tracked continuously by the feature trackers. Also, we can resume updating the color histograms.

Bibliography

[1] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, January 1989.

[2] Benedicte Bascle, Patrick Bouthemy, Rachid Deriche, and Francois Meyer. Tracking complex primitives in an image sequence. In *Proceedings of the 12th International Conference on Pattern Recognition*, pages 426–431, October 1994.

[3] Craig Becker. Personal communication. 1995.

[4] Roberto Brunelli and Tomaso Poggio. Face recognition: Features versus templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):1042–1052, October 1993.

[5] J. F. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):34–43, 1986.

[6] Olivier Faugeras. *Three-Dimensional Computer Vision*, chapter 8, pages 301–340. The MIT Press, 1993.

[7] Arthur Gelb. *Applied Optimal Estimation*, chapter 4, pages 102–143. The MIT Press, 1974.

[8] B. K. P. Horn and B. G. Schunk. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[9] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993.

[10] D. P. Huttenlocher, J. J. Noh, and W. J. Rucklidge. Tracking non-rigid objects in complex scenes. In *Proceedings of the 4th International Conference on Computer Vision*, pages 93–101, May 1993.

[11] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell. Towards robust and automatic traffic scene analysis in real-time. In *Proceedings of the 12th International Conference on Pattern Recognition*, pages 126–131, October 1994.

[12] Jae S. Lim. *Two-Dimensional Signal and Image Processing.* Prentice Hall, 1990.

[13] David G. Lowe. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, 8(2):113–122, 1992.

[14] Pattie Maes, Trevor Darrell, Bruce Blumberg, and Sandy Pentland. The ALIVE system: Full-body interaction with animated autonomous agents. Media Laboratory Perceptual Computing Technical Report 257, Massachusetts Institute of Technology, January 1994.

[15] A. Makarov, J. M. Vesin, and M. Kunt. Intrusion detection using extraction of moving edges. In *Proceedings of the 12th International Conference on Pattern Recognition*, pages 804–807, October 1994.

[16] Sun Microsystems. Starfire: A vision of future computing. Video, 1995.

[17] Steven J. Nowlan and John C. Platt. A convolutional neural network hand tracker. In *Advances in Neural Information Processing Systems 7*, 1995.

[18] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall, 1989.

[19] Satyajit Rao. Personal communication. 1994.

[20] Miguel Schneider. Personal communication. 1994.

[21] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR*, pages 593–600, ??June 1994.

[22] Akio Shio and Jack Sklansky. Segmentation of people in motion. In *IEEE Workshop on Visual Motion*, pages 325–332, 1991.

[23] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[24] Mike Wessler. A modular visual tracking system. Master's thesis, Massachusetts Institute of Technology, 1995.

[25] John Woodfill and Ramin Zabih. An algorithm for real-time tracking of non-rigid objects. In *Proceedings of the American Association for Artificial Intelligence Conference*, pages 718–723, 1991.

2307-5