Pfinder: Real-Time Tracking of the Human Body

by

Christopher R. Wren

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

Masters of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1996

Author . . . .
Department of Electrical Engineering and Computer Science
August 1, 1996

Certified by . . . .
Alex P. Pentland
Associate Professor
Thesis Supervisor

Accepted by . . . .
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Students

Pfinder: Real-Time Tracking of the Human Body

by

Christopher R. Wren

Submitted to the Department of Electrical Engineering and Computer Science
on August 1, 1996, in partial fulfillment of the
requirements for the degree of
Masters of Science in Electrical Engineering and Computer Science

Abstract

Pfinder is a real-time system for tracking and interpretation of people. It runs on a standard SGI Indy computer, and has performed reliably on thousands of people in many different physical locations. The system uses a multi-class statistical model of color and shape to segment a person from a background scene, and then to find and track people's head and hands in a wide range of viewing conditions. Pfinder produces a real-time representation of a user useful for applications such as wireless interfaces, video databases, and low-bandwidth coding, without cumbersome wires or attached sensors.

Thesis Supervisor: Alex P. Pentland
Title: Associate Professor

Acknowledgments

Many people contributed to this thesis in one way or another. I'd like to thank a few of them here:

My thesis supervisor, Sandy Pentland, was critical to the execution of this thesis, providing fertile ground, technical knowledge, and motivation. I've grown a bit, both technically and philosophically, in the past two years, and I'm looking forward to the bright prospects of the future.

My academic advisor, Eric Grimson, and the whole E.E.C.S. department, deserve special credit for their wisdom and iron-clad patience. Special thanks to Marilyn Pierce for her help with all my silly questions.

Around the Lab there are many, many people who deserve acknowledgment, but as always, only a few will find it: Irfan Essa for his guidance and advice, Lee Campbell and Kris Popat for their undying enthusiasm toward solving puzzles and problems, Flavia Sparacino for her creative answers to the question "What's *that* good for?", and Dave Becker for his perspective and love of life.

Thanks also go to Bruce Blumberg and Trevor Darrell for creating the ALIVE project, and to the many who have worked on Pfinder and its ancestor, backsub: Ali Azarbayejani, Michael P. Johnson, Kenneth Russell, and Thad Starner. If it weren't for ALIVE and backsub, there wouldn't be a Pfinder.

My parents deserve kudos for their encouragement, support, and love over the years. Their sacrifices, foresight, and guidance are largely responsible for me being *me*[1].

Finally, I'd like to thank my wonderful wife, Donna, for enduring my admittedly obnoxious graduate-student lifestyle with poise and grace. She's been amazingly loving, supportive and understanding throughout. All my love to you, Donna.

[1] so blame them

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Applications such as video databases, wireless virtual reality interfaces, smart rooms, very-low-bandwidth video compression, and security monitoring all have in common the need to track and interpret human action. The ability to find and follow people's head, hands, and body is therefore an important visual problem.

To address this need we have developed a real-time system called Pfinder ("person finder") that substantially solves the problem for arbitrarily complex but single-person, fixed-camera situations[1]. The system provides interactive performance on general-purpose hardware, and has been tested on thousands of people in several installations around the world, and has performed quite reliably.

Pfinder has been used as a real-time interface device for information spaces[19], performance spaces[22], video games[18], and a distributed virtual reality populated by artificial life[6]. It has also been used as a pre-processor for gesture recognition systems, including one that can recognize a forty-word subset of American Sign Language with near perfect accuracy [20].

Pfinder adopts a Maximum *A Posteriori* Probability (MAP) approach to detection and tracking of the human body using simple $2\frac{1}{2}$-D models. It incorporates *a priori* knowledge about people primarily to bootstrap itself and to recover from errors. The central tracking and description algorithms, however, can equally well be applied to tracking vehicles or animals, and in fact we have done informal experiments in these areas. Pfinder is a descendant of the vision routines originally developed for the ALIVE system [7], which performed person tracking but had no explicit model of the person and required a controlled background. Pfinder is a more general, and more accurate, method for person segmentation, tracking, and interpretation.

1.1 Related Work

Pfinder is descended from a variety of interesting experiments in human-computer interface and computer mediated communication. Initial exploration into this space

[1] Use of existing image-to-image registration techniques [2, 13] allow Pfinder to function in the presence of camera rotation and zoom, but real-time performance cannot be achieved without special-purpose hardware

of applications was by Krueger [11], who showed that even 2-D binary vision processing of the human form can be used as an interesting interface. More recently the Mandala group [1], has commercialized and improved this technology by using analog chromakey video processing to isolate colored gloves, etc., worn by users. In both cases, most of the focus is on improving the graphics interaction, with the visual input processing being at most a secondary concern. Pfinder goes well beyond these systems by providing a detailed level of analysis impossible with primitive binary vision.

Pfinder is also related to body-tracking projects like those by Rehg and Kanade [16], Rohr [17], and Gavrila and Davis [8] that use kinematic models, or those by Pentland and Horowitz [15] and Metaxas and Terzopolous [14] that use dynamic models. Such models, however, require reasonably accurate initialization, a currently unsolved problem. In addition, despite some efforts to handle occlusion, currently such models cannot reliably deal with large occlusions. Finally, such approaches require relatively massive computational resources to run in real-time.

Pfinder is perhaps most closely related to the work of Bichsel [5] and Baumberg and Hogg [4]. These systems segmented the person from the background in real time using only a standard workstation. Their limitation is that they did not analyze the person's shape or internal features, but only the silhouette of the person. Consequently, they cannot track head and hands, recognize any but the simplest gestures, or determine body pose.

1.2 Motivation

Pfinder goes beyond those systems by also building statistical models of the person's clothing, head, hands, and feet. This multi-class approach, as described in Chapter 2, provides not only more robust tracking, but also a richer description of the scene as compared to single-sided classification techniques. The statistical, region-based nature of features allows the tracking to proceed, stably, at interactive speeds, without special purpose hardware. Chapter 4 provides some concrete information about estimation stability and execution performance.

The statistical nature of the tracking algorithm makes it possible to include *a priori* knowledge about the nature of subjects to be tracked. Chapter 3 discusses how this knowledge, as well as heuristic techniques, can be combined to provide automatic, quick, and reliable initialization and error recovery.

Pfinder's performance, and stability, have resulted in it being utilized by a number of whole-body interaction applications. Chapter 5 provides an overview of some of these systems.

Chapter 2

Steady State Tracking

This chapter will describe Pfinder's representations and operation in the "steady-state" case, where it has already found and built a model of the person and scene. Chapter 3 will then describe the initialization or model-building process, and the error recovery process.

2.1 Modeling The Person

Figure 2-1: Analysis of a user in the ALIVE environment. The frame on the left is the video input (n.b. color image shown here in black and white for printing purposes), the center frame shows the segmentation of the user into blobs, and the frame on the right shows a model reconstructed from blob statistics alone (with contour shape ignored).

The human is modeled as a connected set of blobs. Each blob has a spatial (x, y) and color (Y, U, V) Gaussian distribution, and a *support map* that indicates which pixels are members of the blob. We define $\mathbf{m}_k$ to be the mean (x, y, Y, U, V) of blob k, and $\mathbf{K}_k$ to be the covariance of that blob's distribution. Because of their different semantics, the spatial and color distributions are assumed to be independent. That is, $\mathbf{K}_k$ is block-diagonal, with uncoupled spatial and spectral components.

Each blob has associated with it a *support map*, that indicates which image pixels are members of a particular blob. We define $s_k(x, y)$, the support map for blob k, to be

$$s_k(x, y) = \begin{cases} 1 & \text{if pixel (x, y) is a member of class } k \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Figure 2-2: The user's hand and face blobs have very similar color statistics. These two blobs end up very close to each other in this frame.

An aggregate support map $s(x, y)$ over all the blob models is also a useful data structure. Since the individual support maps indicate which image pixels are members of that particular blob, the aggregate support map represents the segmentation of the image into spatial/color classes.

Each blob can also have a detailed representation of its shape and appearance, modeled as differences from the underlying blob statistics. The ability to efficiently compute compact representations of people's appearance is useful for low-bandwidth applications, such as our demonstration of a shared virtual environments at SIGGRAPH '95 [6].

The statistics of each blob are recursively updated to combine information contained in the most recent measurements with knowledge contained in the current class statistics and the priors. Because the detailed dynamics of each blob are unknown, we use approximate models derived from experience with a wide range of users. For instance, blobs that are near the center of mass have substantial inertia, whereas blobs toward the extremities can move much faster.

2.2 Modeling The Scene

We assume that the majority of the time Pfinder will be processing a scene that consists of a relatively static situation such as an office, and a single moving person. Consequently, it is appropriate to use different types of model for the scene and for the person.

We model the scene surrounding the human as a texture surface; each point on the texture surface is associated with a mean color value and a distribution about that mean. Color is expressed in the YUV space. The color distribution of each pixel is modeled with the Gaussian described by a full covariance matrix. Thus, for instance, a fluttering white curtain in front of a black wall will have a color covariance that is very elongated in the luminance direction, but narrow in the chrominance directions.

We define $\mathbf{m}_0$ to be the mean (Y, U, V) of a point on the texture surface, and $\mathbf{K}_0$ to be the covariance of that point's distribution. The spatial position of the point is treated implicitly because, given a particular image pixel at location (x, y), we need only consider the color mean and covariance of the corresponding texture location. In Pfinder the scene texture map is considered to be class zero.

One of the key outputs of Pfinder is an indication of which scene pixels are occluded by the human, and which are visible. This information is critical in low-bandwidth coding (you don't have to code the background), and in the video/graphics compositing required for "augmented reality" applications.

In each frame visible pixels have their statistics recursively updated using a simple adaptive filter.

$$\mathbf{m}_t = \alpha\mathbf{y} + (1-\alpha)\mathbf{m}_{t-1} \tag{2.2}$$

This allows us to compensate for changes in lighting and even for object movement. For instance, if a person moves a book it causes the texture map to change in both the locations where the book was, and where it now is. By tracking the person we can know that these areas, although changed, are still part of the texture model and thus update their statistics to the new value. The updating process is done recursively, and even large changes in illumination can be substantially compensated within two or three seconds.

2.3 The Analysis Loop

Given a person model and a scene model, we can now acquire a new image, interpret it, and update the scene and person models. To accomplish this there are several steps:

1. First, we predict the appearance of the user in the new image using the current state of our model. This is accomplished using a set of Kalman filters with simple Newtonian dynamics that operate on each blob's spatial statistics.

2. Next, for each image pixel we must measure the likelihood that it is a member of each of the blob models and the scene model. Self-shadowing and cast shadows are a particular difficulty in measuring this likelihood.

3. Resolve these pixel-by-pixel likelihoods into a support map, indicating for each pixel whether it is part of one of the blobs or of the background scene. Spatial priors and connectivity constraints are used to accomplish this resolution.

4. Update the statistical models for each blob and for the background scene; also update the dynamic models of the blobs.

Each of these steps will now be described in more detail.

2.3.1 Predict Model Parameters

The first step is to update the spatial model associated with each blob using the blob's dynamic model, to yield the blob's predicted spatial distribution for the current image:

$$\hat{\mathbf{X}}_{[n|n-1]} = \mathbf{\Phi}_{[n-1]}\hat{\mathbf{X}}_{[n-1|n-1]} \tag{2.3}$$

where the estimated state vector $\hat{X}$ includes the blob's position and velocity, the observations Y are the mean spatial coordinates of the blob in the current image, and the filter $\hat{G}$ is constructed assuming simple Newtonian dynamics. Smaller blobs near the person's extremities (e.g., head, hands, and feet) are assumed to have less inertia than the larger blobs that describe the person's body.

2.3.2 Measure Likelihoods For Each Class

For each image pixel we must measure the likelihood that it is a member of each of the blob models and the scene model.

For each pixel in the new image, we define $\mathbf{y}$ to be the vector (x, y, Y, U, V). For each class k (e.g., for each blob and for the corresponding point on the scene texture model) we then measure the log likelihood

$$d_k = -(\mathbf{y} - \mathbf{m}_k)^T \mathbf{K}_k^{-1} (\mathbf{y} - \mathbf{m}_k) - \ln |\mathbf{K}_k| \tag{2.4}$$

Missing or implicit spatial components are assumed to contribute nothing to the membership likelihood.

Shadowing. Self-shadowing and cast shadows are a particular difficulty in measuring the membership likelihoods, however we have found the following approach sufficient to compensate for shadowing. First, we observe that if a pixel is significantly brighter (has a larger Y component) than the class statistics say it should, then we do not need to consider the possibility of shadowing. It is only in the case that the pixel is darker that there is a potential problem.

When the pixel is darker than the class statistics indicate, we therefore normalize the chrominance information by the brightness,

$$U^* = U/Y \tag{2.5}$$
$$V^* = V/Y \tag{2.6}$$

This normalization removes the effect of changes in the overall amount of illumination. For the common illuminants found in an office environment this step has been found to produce a stable chrominance measure despite shadowing.

The log likelihood computation then becomes

$$d_k = -(\mathbf{y}^* - \mathbf{m}_k^*)^T {\mathbf{K}^*}_k^{-1} (\mathbf{y}^* - \mathbf{m}_k^*) - \ln |\mathbf{K}_k^*| \tag{2.7}$$

where $\mathbf{y}^*$ is (x, y, U^*, V^*) for the image pixel at location (x, y), $\mathbf{m}_k^*$ is the mean (x, y, U^*, V^*) of class k and $\mathbf{K^*}_k$ is the corresponding covariance.

2.3.3 Determine Support Map

The next step is to resolve the class membership likelihoods at each pixel into support maps, indicating for each pixel whether it is part of one of the blobs or of the scene. Spatial priors and connectivity constraints are used to accomplish this resolution.

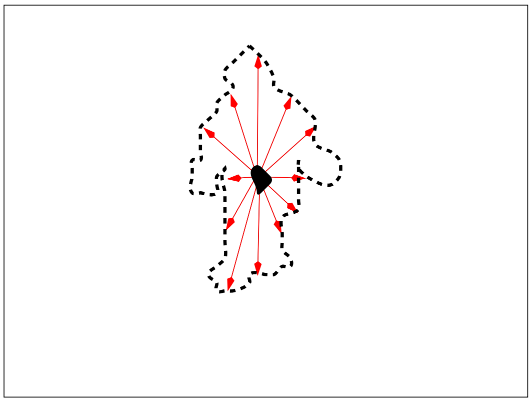


Figure 2-3: The morphological grow algorithm used by Pfinder to insure a connected foreground region

Individual pixels are then assigned to particular classes: either to the scene texture class or a foreground blob. A classification decision is made for each pixel by comparing the computed class membership likelihoods and choosing the best one (in the MAP sense), e.g.,

$$s(x, y) = \arg\max_k (d_k(x, y)) \tag{2.8}$$

Morphology. Connectivity constraints are enforced by iterative morphological "growing" from a single central point, to produce a single region that is guaranteed to be connected (see Fig. 2-3. The first step is to morphologically grow out a "foreground" region using a mixture density comprised of all of the blob classes. This defines a single connected region corresponding to all the parts of the user. Each of the individual blobs are then morphologically grown, with the constraint that they remain confined to the foreground region.

This results in a set of connected blobs that fill out the foreground region. However the boundaries between blobs can still be quite ragged due to misclassification of individual pixels in the interior of the figure. We therefore apply spatial Gaussian smoothing when determining the scene class likelihoods.

2.3.4 Update Models

Given the resolved support map $s(x, y)$, we can now update the statistical models for each blob and for the scene texture model. By comparing the new model parameters to the previous model parameters, we can also update the dynamic models of the blobs.

For each class k, the pixels marked as members of the class are averaged together to produce the model mean $\mathbf{m}_k$, and their second-order statistics measured to calculate the model's covariance matrix $\mathbf{K}_k$,

$$\mathbf{K}_k = E[(\mathbf{y} - \mathbf{m}_k)(\mathbf{y} - \mathbf{m}_k)^T] \tag{2.9}$$

This process can be simplified by re-writing it in another form more conducive to iterative calculation. The first term can be built up as examples are found, and the mean can be subtracted when it is finally known:

$$E[(\mathbf{y}-\mathbf{m}_k)(\mathbf{y}-\mathbf{m}_k)^T] = E[\mathbf{y}\mathbf{y}^T] - \mathbf{m}_k\mathbf{m}_k^T \tag{2.10}$$

For computational efficiency, color models are built in two different color spaces: the standard (Y, U, V) space, and the brightness-normalized (U^*, V^*) color space.

Blending Measurements and Priors. Errors in classification and feature tracking can lead to instability in the model. Special care must be taken to make sure the model remains valid. One way to accomplish this is to reconcile the individual blob models with domain-specific prior knowledge. For instance, some parameters (e.g., color of a person's hand) are expected to be stable and to stay fairly close to the prior distribution, some are expected to be stable but have weak priors (e.g., shirt color) and others are both expected to change quickly and have weak priors (e.g., hand position).

Intelligently chosen prior knowledge can turn a class into a very solid feature tracker. For instance, classes intended to follow flesh are good candidates for assertive prior knowledge, because people's normalized skin color is surprisingly constant across race and tan. There is some variation with very different lighting conditions (e.g., sun versus fluorescent), so a small library of skin classes is required to handle different illumination conditions.

Chapter 3

Initialization and Recovery

Pfinder's initialization process consists primarily of building representations of the person and the surrounding scene. It first builds the scene model by observing the scene without people in it, and then when a human enters the scene it begins to build up a model of that person.

The person model is built by first detecting a large change in the scene, and then building up a multi-blob model of the user over time. The model building process is driven by the distribution of color on the person's body, with blobs being added to account for each differently-colored region. Typically separate blobs are required for the person's hands, head, feet, shirt and pants.

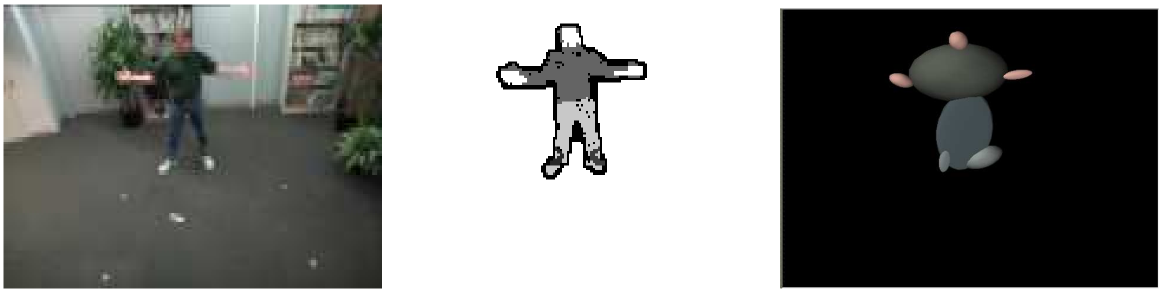


Figure 3-1: Segmentation and model of a user in a "star fish" pose. In this pose it is possible to locate the key body parts (head, hands, feet) using contour analysis alone

The process of building a blob-model is guided by a 2-D contour shape analysis that recognizes silhouettes in which the body parts can be reliably labeled. For instance, when the user faces he camera and extends both arms (what we refer to as the "star fish" configuration, see Fig. 3-1) then we can reliably determine the image location of the head, hands, and feet. When the user points at something, then we can reliably determine the location of the head, one hand, and the feet.

These locations are then integrated into blob-model building process by using them as prior probabilities for blob creation and tracking. For instance, when the face and hand image positions are identified we can set up a strong prior probability for skin-colored blobs.

The following sections describe the blob-model building process in greater detail.

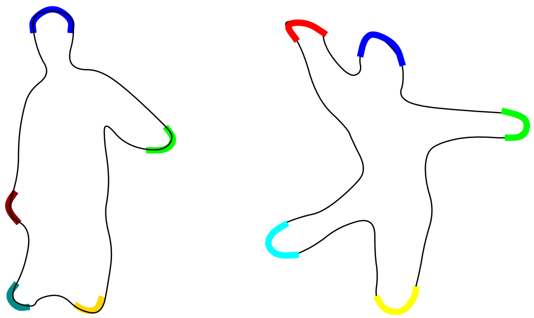


Figure 3-2: To initialize the blob models, Pfinder uses features found by a contour analysis module.

3.1 Learning The Scene

Before the system attempts to locate people in a scene, it must learn the scene. To accomplish this Pfinder begins by acquiring a sequence of video frames that do not contain a person. Typically this sequence is relatively long, a second or more, in order to obtain a good estimate of the color covariance associated with each image pixel. For computational efficiency, color models are built in both the standard (Y, U, V) and brightness-normalized (U^*, V^*) color spaces.

With a static camera (the normal case) the pixels of the input images correspond exactly to the points in the scene texture model. This makes processing very efficient. In order to accommodate camera rotation and zooming, the input image must be transformed back to the coordinate system of the scene texture model before we compare the input image and scene model. Although we can estimate the camera transform parameters in real time [2, 13], we cannot currently apply the transform to the input image in real time.

3.2 Detect Person

After the scene has been modeled, Pfinder watches for large deviations from this model. New pixel values are compared to the known scene by measuring their Mahalanobis distance in color space from the class at the appropriate location in the scene model, as per Equation 2.4.

If a changed region of the image is found that is of sufficient size to rule out unusual camera noise, then Pfinder proceeds to analyze the region in more detail, and begins to build up a blob model of the person.

3.3 Building the Person Model

Modeling and subsequent analysis of the user utilizes the Gaussian blobs described above, incorporating both spatial and color information.

To initialize blob models, a 2D contour shape analysis that attempts to identify the head, hands, and feet locations (see Fig. 3-2). When this contour analysis does identify one of these locations, then a new blob is created and placed at that location. For hand and face locations, the blobs have strong flesh-colored color priors. Other blobs are initialized to cover clothing regions.

The blobs introduced by the contour analysis compete with all the other blobs to describe the data. As a consequence, there are typically only seven blobs required: three for head and hands, two for feet, and one each for shirt and pants.

Occlusion. When a blob can find no data to describe (as when a hand or foot is occluded), it is deleted from the person model. When the hand or foot later reappears, a new blob will be created by either the contour process (the normal case) or the color splitting process. This deletion/addition process makes Pfinder very robust to occlusions and strong shadows. When a hand reappears after being occluded or shadowed, normally only a few frames of video will go by before the person model is again accurate and complete.

3.3.1 Contour Analysis

Much of the time a person's head, feet and hands extend out from the projected shape of their body, forming very salient features in the contour of their silhouette. Consequently, once the initial figure-ground segmentation has been obtained, it is possible to identify these features and thus reliably locate the person's head, hands, and feet.

This is accomplished by "walking" around the edge of the segmented foreground region, producing a chain code model of the person's 2D silhouette, and identifying the large protrusions. The geometry of these protrusions is then analyzed using statistically-derived rules to determine which protrusion is the head, which the hands, and so forth [10, 7]. This contour chain-code is also very useful for some applications, because it is a much more compact representation of the person than the segmentation bitmap.

3.3.2 Integrating of Blob and Contour Features

The blob models and the contour analyzer produce many of the same features (head, hands, feet), but with very different failure modes. The contour analysis can find the features in a single frame if they exist, but the results tend to be noisy. The class analysis produces accurate results, and can track the features where the contour can not, but it depends on the stability of the underlying models and the continuity of the underlying features (i.e., no occlusion).

The last stage of model building involves the reconciliation of these two modes. For each feature, Pfinder heuristically rates the validity of the signal from each mode. The signals are then blended with prior probabilities derived from these ratings. This allows the color trackers to track the hands in front of the body—when the hands produce no evidence in the contour. If the class models become lost due to occlusion or rapid motion, the contour tracker will dominate and will set the feature positions once they are re-acquired in the contour.

Chapter 4

Performance

Two measures of performance are relevant to Pfinder. The first is classification performance. This is a measure of how well Pfinder is classifying the scene, and will be examined below by measuring the error and noise in the estimates of body-part positions that are based on the classification output.

Since Pfinder is intended to be an interactive system, the other important measure is the runtime performance. This is measured below in terms of per-frame and per-pixel computation rates.

4.1 Classification Performance

Pfinder usually obtains video signal from a JVC-1280C, single CCD, color camera. It provides an S-video signal to the SGI digitizers. Its specs claim 40db signal to noise. The SGI digitizer smoothes and sub-samples the images down to sixteenth resolution. Give this input, Pfinder exhibits Root Mean Squared (RMS) tracking errors on the order of a few pixels in position and a few degrees in orientation, as shown in Table 4.1. Here, the term “hand” refers to the region from approximately the wrist to the fingers. An “arm” extends from the elbow to the fingers.

For the translation tests, the user moves through the environment while holding onto a straight guide. Absolute errors (less then 6 pixels for the hand, and 22 pixels for the arm) are measured from the reported position to the position of the guide path in the image. Residual errors (sub-pixel for the hand, and little more than 2 pixels for the arm) ignore errors intrinsic to the experimental setup: the tracked region is constrained to move in a straight line in the image, but nothing constrains the center of the tracked area to be coincident with the center of the guide. Note that these accuracies are obtained from images that are sub-sampled by a factor of four in each dimension. Relative error is the ratio of the RMS error to the total screen size. Figure 4-1 shows some sample results from the trials.

All plots if Figure 4-1 show reported X position versus reported Y position. Plot 4-1a shows data from tracking the left hand of a long-sleeved subject, so the tracked region extended from the wrist to the fingers. Plot 4-1b shows data from a similar run on the subject's right hand. Plot 4-1c shows data from tracking the left arm of

test	hand	arm
position		
(X,Y) absolute	5.8 pixels (0.9% rel)	22.0 pixels (3.4% rel)
(X,Y) residual	0.7 pixels (0.1% rel)	2.1 pixels (3.2% rel)
orientation		
(Θ) residual	4.8 degrees (5.2% rel)	3.0 degrees (3.1% rel)

Table 4.1:
Absolute position error is the RMS error between the reported position and the guide path. Residual position error the RMS error after model errors accounted for. Orientation error is the RMS error to a low-passed filtered version of the data since no ground truth was available. "rel" designates relative error: the ratio of the error to the total path length.

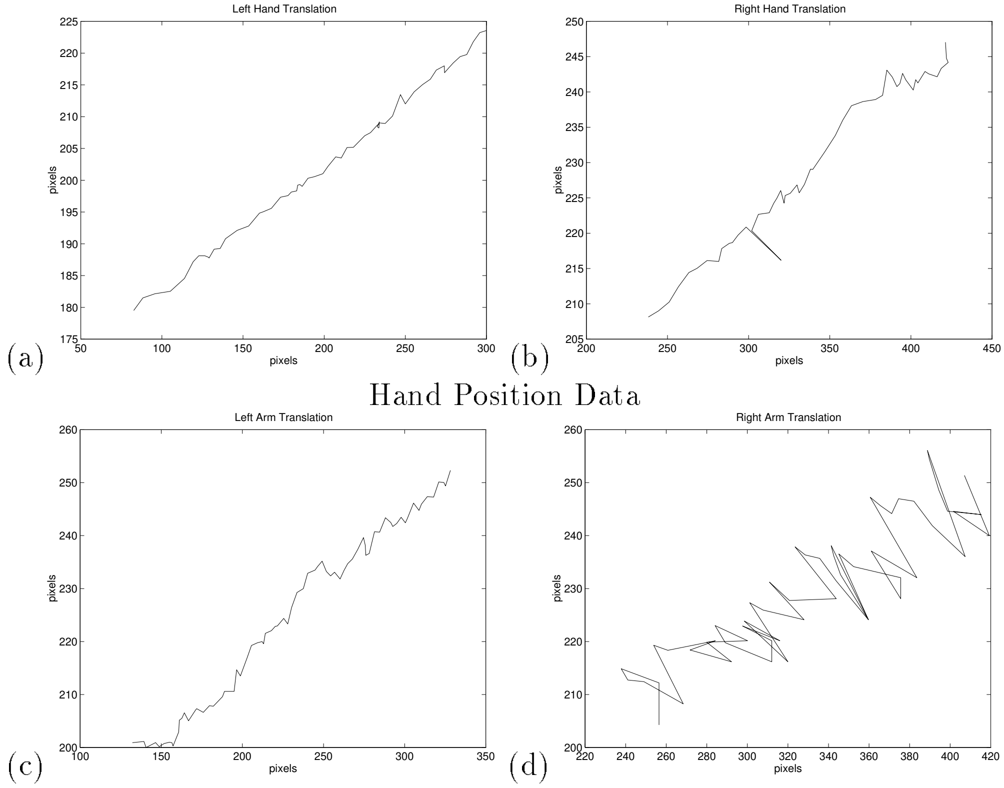
a short-sleeved subject. In this case the tracked region extended from approximately the elbow to the fingers. Plot 4-1d shows data from a similar run on the subject's right arm. This run exhibits some instability, probably due to a combination of factors including transient partial occlusions of the arm.

For the orientation error test, the user smoothly moves an appendage through several cycles of approximately 90 degree rotation. There is no guide in this test, so neither the path of the rotation, nor even its absolute extent, can be used to directly measure error. The RMS distance to a low-pass filtered version of the data provides a measure of the noise in the data. Figure 4-2 shows some data from the orientation test.

All plots if Figure 4-2 show reported Θ position versus time. Plot 4-2a shows data from tracking the left hand of a long-sleeved subject, again the tracked region extended from the wrist to the fingers. Plot 4-2b shows data from a similar run on the subject's right hand. Plot 4-2c shows data from tracking the left arm of a short-sleeved subject. As in the position tests the tracked arm region extended from the elbow to the fingers. Plot 4-2d shows data from a similar run on the subject's right arm. The hand orientation data exhibits more noise. This likely due to the fact that the tracked region in the arm case has significantly more eccentricity than in the hand case. That extra information allows the model blobs to be more stable in orientation.

4.2 Runtime Performance

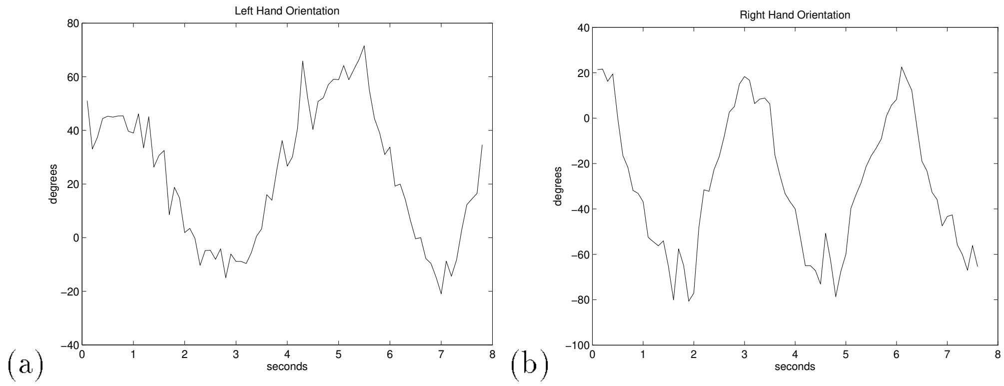
Pfinder is implemented on the SGI architecture using the VL (Video Library) interface. A typical frame-rate on sixteenth resolution (160x120 pixel) frames is 10Hz using a 200MHz R4400 processor Indy with Vino Video. The same code also runs on the Indigo^2 platform with Galileo video, but is substantially slower. The difference in speed seems to derive from the fact that the Indy moves video into DRAM with little CPU intervention.



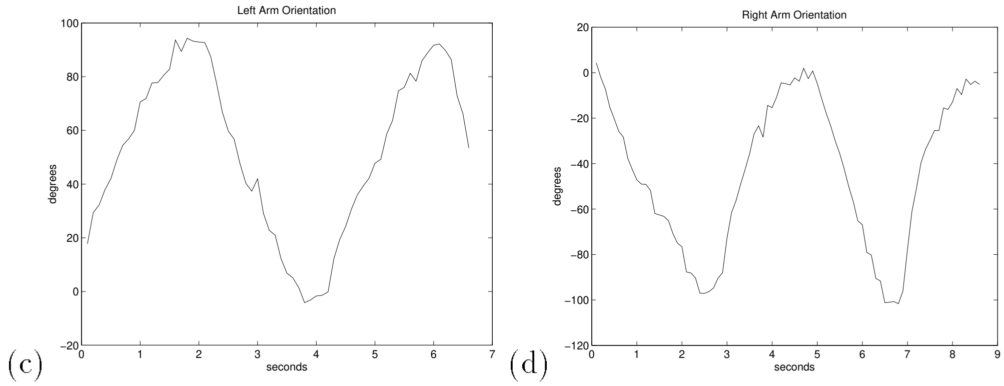
Arm Position Data

Figure 4-1:
These figures show example data from of independent runs of tracking while the hand was slid along a straight guide. **(a)** is the result of tracking the left hand. **(b)** is the result of tracking the right hand. **(c)** is the result of tracking the left arm. **(d)** is the result of tracking the right arm. **Hand** refers to the visible flesh area when a person wears a long sleeve shirt. **Arm** refers to the visible flesh when a person wears a short sleeve shirt.

The user model allows for an optimization called *grow*ing. The grow operation is allows Pfinder to touch as few uninteresting pixels as possible. By starting at model-determined seed points and expanding analysis outward, Pfinder identifies all the pixels the belong to a single connected region (given the assumption that people are connected). The algorithm for the grow is the familiar "brush fire" algorithm from computer graphics work[9]. Although the grow is much less efficient on a per-pixel basis, it provides an overall performance gain, as shown in Table 4.2.



Hand Orientation Data



Arm Orientation Data

Figure 4-2:
These figures show the results of independent runs of an experiment designed to test stability of orientation tracking. No guide was used, the subject attempted to move the specified appendage smoothly through several iterations of 90 degrees. **(a)** is the result of tracking the left hand. **(b)** is the result of tracking the right hand. **(c)** is the result of tracking the left arm. **(d)** is the result of tracking the right arm. **Hand** refers to the visible flesh area when a person wears a long sleeve shirt. **Arm** refers to the visible flesh when a person wears a short sleeve shirt.

	grow	raster
frame rate	10.2 Hz	3.2 Hz
per pixel time	119 μsec	16 μsec

Table 4.2:
Gross frame rates and normalized per-pixel computation times for the 'grow' method versus a raster scan approach. The grow code is much less efficient on a pixel-by-pixel basis, but it increases overall performance.

Chapter 5

Applications

Although interesting by itself, the full implications of real-time human tracking only become concrete when the information is used to create an interactive application. Pfinder has been used to explore several different human interface applications. This section describes some systems that have been developed in conjunction with fellow researchers. They are presented here to help illustrate the potential of Pfinder as an interface device.

Pfinder provides a modular interface to client applications. Several clients can be serviced in parallel, and clients can attach and detach without affecting the underlying vision routines. A wide range of data is exported through this interface:

- polygon representation of the support map
- blob model statistics
- semantically labeled feature positions (e.g. head, right hand, etc.)
- gestures (e.g. standing, sitting, pointing, etc.)

Before this information can be useful for applications, a mapping must exist between Pfinder output and the interface idiom. This mapping must be carefully chosen, because it defines the metaphor that the user is forced use when they interact with the system. The desired level of abstraction, tolerance to interface accuracy and lag, even the prior expectations of the user must be taken into account when designing this mapping.

5.1 SURVIVE

The simplest mapping is, of course, the direct one: map interface device features directly (one-to-one) into the control space of some application. Usually a small amount of filtering will be required, and possibly it's desirable to use non-linear mappings, but otherwise interface outputs feed directly into application inputs.

Russell and Starner[18] created an application called SURVIVE (Simulated Urban Recreational Violence Interactive Virtual Environment), an entertainment application

(a) 

Figure 5-1: The user environment for SURVIVE.

that uses a direct mapping. SURVIVE allows the user to interact with a 3D game environment using the IVE space. Figure 5-1 shows a user in SURVIVE. The gestural interpretation provided by Pfinder is mapped into the game controls for the popular id Software game Doom.

It's insightful to contrast the SURVIVE interface with the standard keyboard Doom interface. The task in Doom is navigating through a virtual environment. This is usually accomplished by pressing keys on a keyboard. Changing the direction of travel is as easy as picking up one finger and pressing down another. Split-second decisions become split-second actions. The SURVIVE interface is much less forgiving. Movement of the virtual body is linked to the movement the real body. A change of virtual direction actually requires a movement in that direction, maybe several feet of movement. This leads to a much more engrossing, visceral experience of the game.

5.2 Visually-Animated Characters

Figure 5-2:
A synthetic character taking direction from a human user who is being tracked in 3-D with stereo vision

A literal mapping is one that treats the tracking features as exactly what they are: evidence about the physical configuration of the user in the real world. In this context the tracking information becomes useful for understanding simple pointing gestures. With quite a bit more work, systems can use this information to estimate a more complete picture of the user's configuration.

Azarbayejani and Pentland[3] are currently building a system which combines Pfinder-based monocular tracking systems into a wide-baseline stereo system called the STereo Interactive Virtual Environment (STIVE). STIVE is capable of resolving 3-D position and orientation from the given 2-D position and orientation tracking.

Wren and Pentland have recently combined STIVE with a literal mapping between user configuration and corresponding parts of an animated character to create an animation-by-example system. The system allows the user to animate the upper body movements of a virtual puppet by executing the corresponding motions (see Figure 5-2). The features from the vision system drive a dynamic human-body model inside the puppet.

5.3 NetSpace

(a) (b)

Figure 5-3:
(a) User browsing the web in NetSpace **(b)** NetSpace landscape with some of the authors' web pages

A gesture-based interface mapping interposes a layer of pattern recognition between the input features and the application control. When an application has a discrete control space, this mapping allows patterns in feature space, better known as gestures, to be mapped to the discrete inputs. The set of patterns form a gesture-language that the user must learn. It is worth noting that this kind of rigid gesture-language tends to be sensitive to failures in tracking, classification, and user training. Systems that employ this kind of mapping must have very flexible, and quick, mechanisms for resolving misunderstandings. See Sections 5.5 and 5.6, for interesting answers to this problem. Sparacino and Hlavac's[22] Netspace is an example of an application that uses a gesture-based mapping.

NetSpace is an immersive, interactive web browser that makes use of people's strength at remembering the surrounding 3D spatial layout. NetSpace maps the

Figure 5-4:
Real-time reading of American Sign Language (with Thad Starner doing the signing).

contents of URLs into a 3D graphical world projected on the large IVE screen. This gives the user a sense the URLs existing in a surrounding 3D environment. To navigate this virtual 3D environment, users stand in front of the screen and use voice and hand gestures to explore (Figure 5-3).

5.4 American Sign Language

Starner and Pentland[20] explored the extremes of gesture-based mapping by combining th pfinder statistics representation with hidden Markov modeling to interpret a forty word subset of American Sign Language (ASL). Using this approach they were able to produce a real-time ASL interpreter with a 99% sign recognition accuracy. Thad Starner is shown using this system in Fig. 5-4.

5.5 DanceSpace

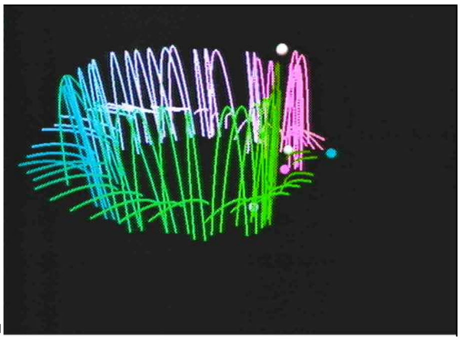
(a)  (b) 

Figure 5-5: **(a)** User dancing with her colored shadow in DanceSpace **(b)** Dancing shadow generated by the user in DanceSpace

Closely related to the gesture-based interface mapping discussed in Section 5.3, the conductor-style interface mapping of Sparacino's[22] DanceSpace also uses a form of predefined gesture language. Unlike the rigid gesture-language of NetSpace the conductor mapping results in a much more fluid interface. The system is designed to produce constructive, interesting results the user doesn't know the details of the mapping.

DanceSpace is an interactive performance space where both professional and non-professional dancers can generate music and graphics through their body movements (See Figure 5-5).

5.6 ALIVE

Figure 5-6: Chris Wren playing with Bruce Blumberg's virtual dog in the ALIVE space

The last of the gesture-language mappings is the most abstract. Again, it's related to the other gesture-languages discussed above, and the primary distinction lies in a subtle, but important, difference in the design of the interface. Best called "gesture in context" this mapping attempts to create an interface that is intuitive given the context. Ideally, the mapping is aligned so that failures in tracking or classification are transparent to the user. Clever mapping design can thus greatly reduce the need for sensor systems to perform flawlessly by playing off the expectations and socialization of the user. Because of that trait, this was the first system to be implemented in our lab (by Maes, Darrell, Blumberg, and Pentland[12]), in the form of the Artificial Life Interactive Virtual Environment (ALIVE).

ALIVE combines autonomous agents with an interactive space. The user experiences the agents (including hamster-like creatures, a puppet, and a well-mannered dog—Figure 5-6) through a "magic-mirror" idiom. The interactive space mirrors the real space on the other side of the projection display, and augments that reflected reality with the graphical representation of the agents and their world (including a water dish, partitions, and even a fire hydrant).

Chapter 6

Future Work

Current applications make use of the information that Pfinder produces by making implicit assumptions about how the that information relates to the state of the user. Wren and Pentland[23] have begun work on a system that utilizes dynamic and stochastic components to explicitly model the human body.

The first version of the system assumed a 2-D dynamic model of the user. Figure 6-1 shows several frames from a 5-link, 2-D model as it was interactively driven by a use through Pfinder. Information from Pfinder determines the potential field that is applied to the model. The model filters these influences through the system dynamics to arrive at the lowest energy solution. This solution becomes the new pose estimate. This version obviously suffers from an over simplified body model.

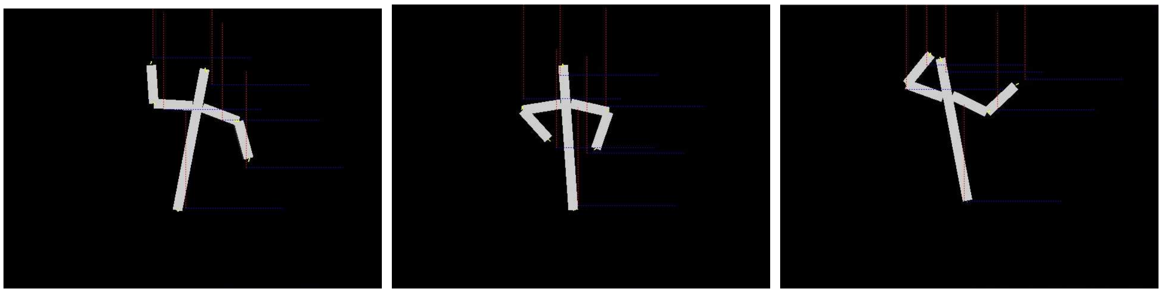


Figure 6-1: The 2-D estimate of the user's upper-body state given Pfinder vision input

The next step was to extend the modeling system to 3-D . Figure 6-2 shows a 5-link, 3-D model being driven by a user through STIVE (a wide-baseline stereo system built on Pfinder technology, see Section 5.2). This model is reacting to the combination of several potential fields: 3-D head and hand positions from STIVE, gravity, and a behavioral prior that affects elbow placement.

This last potential field is particularly interesting. It's a crude attempt at coding behavioral priors in the form of a potential field. With only the STIVE input and gravity, the model places the elbows too close to the body, compared to the user. Solving for the minimum energy in this potential field yields the wrong answer because the user is constrained by more than just physics; the user also has many habits that constrain their motion. Future work in this direction involves codifying those habits

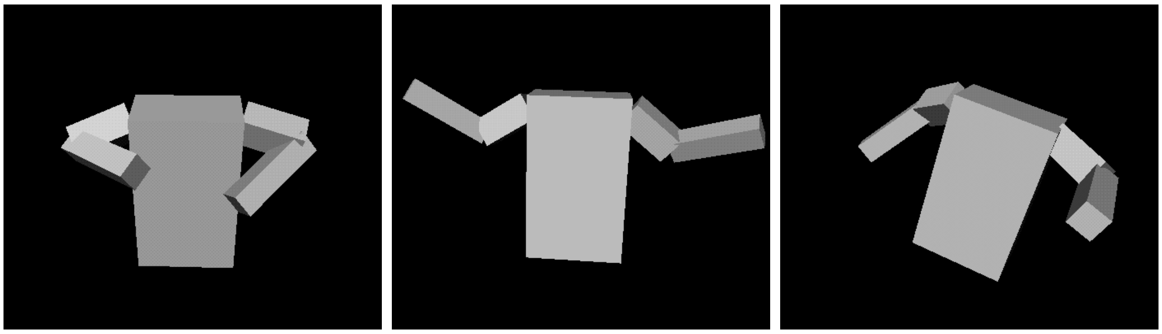


Figure 6-2:
The 3-D estimate of the user's upper-body state given STIVE vision input (STIVE is a Pfinder-based, wide-baseline stereo system, See Section 5.2)

statistically so they can be used to better estimate body pose from 3-D vision data. Eventually, this knowledge may be useful in estimating 3-D body pose from 2-D vision data.

Appendix A

Multi-Class Classification

Classification is the process of sorting feature vectors into categories, or classes. Feature vectors are points in a feature space that is defined to be some collection of measurements, either raw or pre-processed. The process of transforming feature vectors into classification tags is a richly studied topic known as pattern recognition. A brief description of the main ideas used by Pfinder follows, but a very thorough discussion of this material can be found in Therrien[21].

Video chroma-key segmentation is an instructive place to start. Chroma-keying is the process of identifying pixels in an image sequence that are of a particular color, usually for the purpose of compositing two video signals. The classes for chroma-keying are foreground and background. The features are raw video pixel values. The only thing the keyer models is the color of the background pixels. The crudest (and least effective) keyer would simply compare each pixel in the frame with the target color and label them with the with the result of the comparison: equality indicates a background pixel, inequality indicates a foreground pixel.

Since there is likely to be noise in the video signal, the crude approach is doomed to failure. The keyer must assume a certain neighborhood, in color space, around the target color that must be classified as background along with the target color. Classification then involves computing some distance to the target color and comparing that distance to a tunable threshold: less than threshold indicates background, greater indicates foreground.

The distribution of the noise is unlikely to be isotropic in the feature space. A more general keyer might model the distribution of noise and compare distances in a normalized space, instead of the somewhat arbitrary feature space. This is the case illustrated in Figure A-1. The mean is the target background color. The concentric ellipses represent equidistant contours in the normalized space. The threshold is a tunable parameter that moves the decision boundary closer or farther away from the mean. All points inside the decision boundary are labeled as belonging to the class. In the case of the keyer, pixels in this region of feature space are labeled as background pixels.

If the noise model is Gaussian, with mean $\mathbf{m}$, and covariance $\mathbf{K}$, then the normalized distance measure is called the Mahalanobis distance. Give a measurement $\mathbf{y}$,

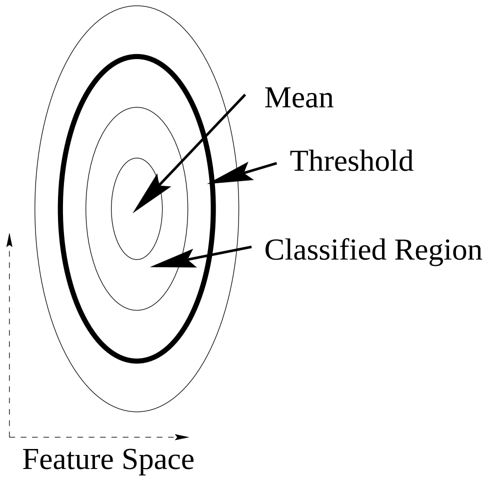


Figure A-1:
Single-sided classification in the 2-D case. The concentric ellipses represent lines of equal probability away from the mean.

this distance can be computed with the following equation[1]:

$$d(\mathbf{y}) = (\mathbf{y} - \mathbf{m})^T \mathbf{K}^{-1} (\mathbf{y} - \mathbf{m}) \tag{A.1}$$

The mean, $\mathbf{m}$, of the Gaussian is the target color in the chroma-key example, and the covariance, $\mathbf{K}$, describes the first-order distribution of the noise.

The main problem with this approach is the threshold. Given a detailed model of the noise, and a desired level of classification performance, it is possible to analytically pick values for the threshold. In practice such detailed models are rare, and are not necessarily stationary. At best, the threshold must be chosen through trial-and-error. At worst, it must be retuned often. Our experience with single-sided classification techniques for person/room segmentation showed that, due to lighting and shadowing, it was necessary to retune the threshold on a frame-to-frame basis. This is an unacceptable situation.

Fortunately, there is well-behaved, analytic solution to the threshold problem: use more than one class. If the task is to separate foreground from background, then model both classes. To classify a measurement $\mathbf{y}$, calculate the distance to each class and choose the smaller distance:

$$\frac{d_A(\mathbf{y})}{d_B(\mathbf{y})} \underset{A}{\overset{B}{\gtrless}} 1 \tag{A.2}$$

[1]The distance metric used in Chapter 2 is the log-likelihood, and is closely related to Mahalanobis distance, with the addition of a negative sign. With the negative sign, less negative (a.k.a. greater) numbers represent points closer to the class. This makes log-likelihood hard to think of as a "distance", so decided to use Mahalanobis distance in this discussion.

The decision boundary that results from this process is the line of equi-probability between the two classes. The two-class situation is illustrated in Figure A-2.

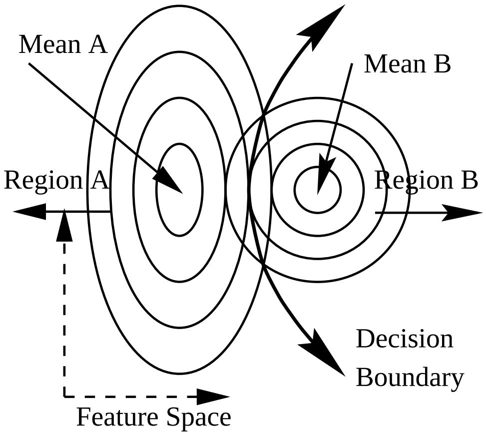


Figure A-2:
Double-sided classification in the 2-D case. The concentric ellipses represent lines of equal probability away from the means. The decision boundary lies where the equal probability lines cross.

Extension to more than two classes is straightforward. The distance to each class is computed, and the classes with the smallest distance labels the pixel:

$$\text{class} = \arg\min_i d_i(\mathbf{y}) \tag{A.3}$$

This is the situation inside Pfinder as described in Chapter 2.

Our experience shows that two-class classification results in better segmentation than single-sided classification. This is the case even when the foreground isn't well modeled by a single Gaussian distribution in color space, because the foreground is a person wearing blue jeans and a white shirt. The single-sided classification is essentially a two-class decision between a Gaussian and a uniform distribution. Even if the foreground class has a large variance, it still contains more information than the uniform distribution, and this leads to better decisions.

Appendix B

Color Spaces

The choice of representation for the color-space can have important consequences for a classification system. As always, a properly chosen representation can make certain operations easier. When tracking a person in a room, it is often necessary to eliminate shadows caused by white, or nearly white, lights. Choosing a color-space representation that makes this easy is a good thing.

There are many color spaces to chose from, each with their own special strengths. However, video digitization hardware tends to provide only a limited selection of formats, and since applying a transform to each pixel is very expensive, the only real choice is usually between RGB, and YUV. The relationship between these two color-spaces is the linear transform described by Equation B.1:

$$\begin{bmatrix} Y \\ U \\ V \\ 1 \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 & 0.063 \\ -0.148 & -0.291 & 0.439 & 0.500 \\ 0.439 & -0.368 & -0.072 & 0.500 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \\ 1 \end{bmatrix} \tag{B.1}$$

However, the important differences between RGB and YUV are best illustrated by Figure B-1. By transforming an RGB color cube into YUV space, it is easy to see that the luma, or brightness, component Y, is orthogonal to the chroma, or color, components U and V. As described in Section 2.3.2, normalization of shadows involves projecting pixel values onto a luma-invariant plane in color-space. In YUV space, normalizing for a white luminant is accomplished simply by discarding the Y component.

Flesh tracking is another operation that the YUV color-space makes easier. If classification is done in the luma-invariant subspace, then a class trained on even an unrepresentative sample population will reliably track flesh across a wide range of skin tones. This convenient outcome derives from the fact that skin pigmentation is always the same color. Varying concentrations only cause variance in luminance.

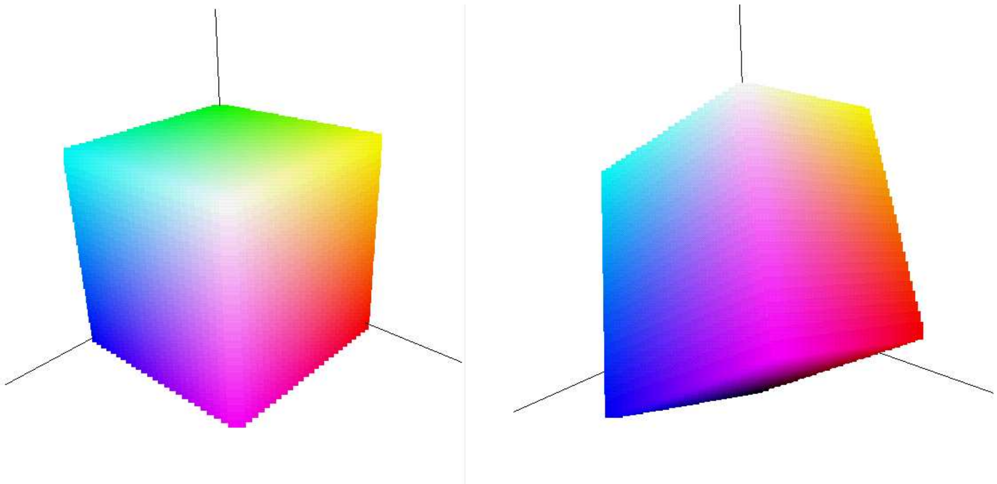


Figure B-1:
The left images shows an RGB color cube in RGB color space. The vertical axis is green (G). The right image shows the same cube transformed into YUV space where the vertical axis is Y (luma). Decisions to based only on color (chroma), can be projected into the U-V plane simply by discarding the Y coordinate.

Bibliography

[1] ACM. *Mandala: Virtual Village*, ACM SIGGraph, Computer Graphics Visual Proceedings, 1993.

[2] A. Azarbayejani and A.P. Pentland. Recursive estimation of motion, structure, and focal length. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(6):562–575, June 1995.

[3] Ali Azarbayejani and Alex Pentland. Real-time self-calibrating stereo person tracking using 3-D shape estimation from blob features. In *Proceedings of 13th ICPR*, Vienna, Austria, August 1996. IEEE Computer Society Press.

[4] A. Baumberg and D. Hogg. An efficient method for contour tracking using active shape models. In *Proceeding of the Workshop on Motion of Nonrigid and Articulated Objects*. IEEE Computer Society, 1994.

[5] Martin Bichsel. Segmenting simply connected moving objects in a static scene. *Pattern Analysis and Machine Intelligence*, 16(11):1138–1142, Nov 1994.

[6] Trevor Darrell, Bruce Blumberg, Sharon Daniel, Brad Rhodes, Pattie Maes, and Alex Pentland. Alive: Dreams and illusions. In *ACM SIGGraph, Computer Graphics Visual Proceedings*, July 1995.

[7] Trevor Darrell, Pattie Maes, Bruce Blumberg, and Alex Pentland. A novel environment for situated vision and behavior. In *Proc. of CVPR–94 Workshop for Visual Behaviors*, pages 68–72, Seattle, Washington, June 1994.

[8] D. M. Gavrila and L. S. Davis. Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. In *International Workshop on Automatic Face- and Gesture-Recognition*. IEEE Computer Society, 1995. Zurich.

[9] Paul S. Heckbert. A seed fill algorithm. In Andrew S. Glassner, editor, *Graphics Gems*, chapter 4, pages 275–277. Academic Press Professional, 1990.

[10] Michael Johnson, Trevor Darrell, and Pattie Maes. Evolving visual routines. *Artificial Life*, 1(4), 1994.

[11] M. W. Krueger. *Artificial Reality II*. Addison Wesley, 1990.

[12] Pattie Maes, Bruce Blumberg, Trevor Darrell, and Alex Pentland. The alive system: Full-body interaction with animated autonomous agents. *ACM Multimedia Systems*, 5:105–112, 1997.

[13] S. Mann and R. W. Picard. Video orbits: characterizing the coordinate transformation between two images using the projective group. *IEEE T. Image Proc.*, 1997. To appear.

[14] D. Metaxas and D. Terzopoulos. Shape and non-rigid motion estimation through physics-based synthesis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15:580–591, 1993.

[15] A. Pentland and B. Horowitz. Recovery of nonrigid motion and structure. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(7):730–742, July 1991.

[16] J.M. Rehg and T. Kanade. Visual tracking of high dof articulated structures: An application to human hand tracking. In *European Conference on Computer Vision*, pages B:35–46, 1994.

[17] K. Rohr. Towards model-based recognition of human movements in image sequences. *CVGIP: Image Understanding*, 59(1):94–115, Jan 1994.

[18] Kenneth Russell, Thad Starner, and Alex Pentland. Unencumbered virtual environments. In *IJCAI-95 Workshop on Entertainment and AI/Alife*, 1995.

[19] Flavia Sparacino, Christopher Wren, Alex Pentland, and Glorianna Davenport. Hyperplex: a world of 3d interactive digital movies. In *IJCAI-95 Workshop on Entertainment and AI/Alife*, 1995.

[20] Thad Starner and Alex Pentland. Real-time american sign language recognition from video using hidden markov models. In *Proceedings of International Symposium on Computer Vision*, Coral Gables, FL, USA, 1995. IEEE Computer Society Press.

[21] Charles W. Therrien. *Decision, Estimation, and Classification.* John Wiley and Sons, Inc., 1989.

[22] C. Wren, F. Sparacino, A. Azarbayejani, T. Darrell, T. Starner, Kotani A, C. Chao, M. Hlavac, K. Russell, and Pentland A. Perceptive spaces for peformance and entertainment: Untethered interaction using computer vision and audition. *Applied Artificial Intelligence*, 11(4):267–284, June 1997.

[23] Christopher Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.