

3 System Description

This section describes the implementation of the system. After the description of each component there is a discussion of the design. The last section of the chapter describes some specific details about the implementation, including the values of many important parameters, and a discussion of the sensitivity of system performance to them.

3.1 Overview

A camera placed below the screen (Figure 1) gives a sequence of images like those in Figure 2. The hand is segmented from low resolution decimations of the image sequence using color and various image processing operations. The size and location of the hand in each image are used to create a sequence of X-location, Y-location, Size (XYS) tokens, which is translated to screen coordinates, smoothed and used to position the cursor where the user is pointing on the screen. The motion of the hand is interpreted by extracting

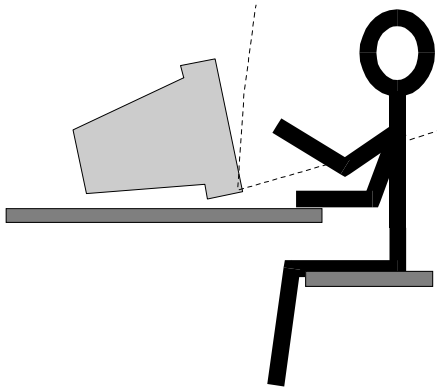


Figure 1: Physical layout.



Figure 2: System's view of user.

symbolic features, such as pauses, changes of direction or distance, from the sequence of smoothed tokens. The sequence of features is interpreted by traversing a transition network encoding the interaction language. At certain points in the network it becomes necessary to classify the pose of the hand. Then a high resolution image is cropped tightly around the region indicated by the current XYS token and a more accurate segmentation is performed. This image is preprocessed to a canonical form and passed to a neural net that has been trained to differentiate the various hand poses. The classification is used to determine the next node in the transition network. At key points in the interpretation of a gesture, the transition network calls out actions that do such things as bring up menus, select or move objects on the screen.

The system is divided into two layers. The lower level runs on a digital signal processor on an image processing board within the host PC producing the XYS tokens. On request from the host it will either produce a token or a close-up of the hand preprocessed for pose recognition. The high level, running on the host, examines the token stream for motion features, runs the pose classification neural nets, navigates through the interaction language and manages the user interface.

3.1.1 Design Discussion

Camera Placement

The camera was placed at the bottom of the screen rather than the top because it provides a much better point of view for observing both location and pose of the user's hand. The user's elbows are below the screen so when the user moves their hands toward the screen the forearm slopes upward. Due this geometry, if the hand were viewed from the top of the screen it would move relatively little in the image whether the user is pointing to the top or bottom of the screen, making vertical positioning less precise. The user is also naturally inclined to allow the hand to align with the forearm during both pointing and non-pointing poses, rather than angle it upwards. Having the camera at the top of the screen would cause the image of the hand to undergo extreme foreshortening, making pose recognition much more difficult. Another consideration is that from above, the hands would always be visible on the keyboard, making it harder to detect when they raise off the keyboard to gesticulate.

There are, however, several disadvantages to having the camera below the monitor. The background contains an array of confusing objects such as ceiling lights, skin-tone doors, the user's face and other people in the background. If the user has bare forearms,

they are more of a problem because they loom larger in the image. These issues have caused some difficulties for segmentation, however given a single camera, any placement is a compromise. Placing the camera below the screen provides the most manageable set of problems.

One factor that may prove to be more important in any future development work is that the typical placement of camera for video-conferencing is above the screen. While a camera below the screen could be used, the view from above is more natural for transmitting faces. For both applications the ideal camera placement is actually *inside* the screen. For video-conferencing this gives an image where the user appears to be looking at you rather than at a point above, below or off to the side. For this application of gesture recognition, a camera view from inside the screen is also desirable because users intuitively present their gestures to the observer, and it is very natural to think of the screen as the observer.

Division of labor between DSP and host

The division of labor between the image processing board and the host was driven largely by two factors, the low bandwidth between that board and host memory, and the relatively low clock speed of the board used. The low communication bandwidth and the need for real-time response forced us to perform all image preprocessing on the DSP board, so that what needs to be passed up to the host is either a small XY token, or a relatively small hand image.

The host must clearly manage the interface to the window system. The open question was whether the gesture interpretation, that is the motion path interpretation, the pose classification and the gesture interpretation should be done on the host or the DSP. This was decided by the relatively slow clock speed of the DSP relative to the host and the relative difficulty of programming it.

It is interesting to note that this division of labor is similar to that of the human brain, where the hippocampus extracts the “where” and the cortex extracts the “what”.

3.2 Hand Segmentation

The hand segmentation process forms the bottom of the information “food chain”, and so must be fast, reliable, consistent, and produce the best quality output possible given the constraints. To support this work, it must be able to run in two modes: a fast

segmentation suitable to identify the location of the hand reliably in real-time; and a high quality segmentation, which can take slightly longer, but which must produce an image suitable to recognize the pose of the hand. This image must be cropped closely around the hand and must contain the majority of the hand and all features needed for pose recognition, but as little background as possible.

3.2.1 Overview

Human skin has a relatively distinct color, rich in lightly saturated red tones. This characteristic persists across a wide range of skin colors and lighting conditions. While the color characteristics are not completely unique, they are sufficiently different from most objects in the environment to make a good starting point for segmentation.

The desired colors are identified using a structure called a Color Predicate (CP). Similar to a Hue/Saturation/Intensity histogram, a CP is a tessellation of color space where cells have a binary value indicating they are either a member of the desired range of colors or not. The CP is trained interactively, then used to identify candidate skin pixels in subsequent images (Figure 3a). Large dense regions of candidate pixels are isolated and used as a mask for the hand (Figure 3b). The color predicate training algorithm is described in Section 3.2.2, and the segmentation process is described in Section 3.2.3.

In addition to this application, the color predicate has been used for a number of other segmentation tasks, including finding face candidates in video sequences and as a tool to help users crop objects from arbitrary images.



Figure 3: (a) Image labeled by CP and (b) the largest connected component.

3.2.2 Color Predicate Training

The predicate is trained using pre-segmented images, which are generated by the user in real time as described at the end of this section. First, pixels with large and small intensities are discarded as their hue and saturation values tend to be unstable [Ke76]. The remaining pixels update the predicate in a manner similar to a histogram. The HSI value of pixels inside the hand region (that is, pixels that presumably have the desired color) are used to identify a target cell in the CP. The cells in a neighborhood around the target are incremented by a Gaussian-weighted value (Figure 4). Likewise, pixels in the background decrease cell counts, but generally use a smaller and narrower weight distribution. The result is a “histogram” of the training images containing both positive and negative values (Figure 8). Finally, to speed run-time performance, cells are thresholded to create the binary color predicate.

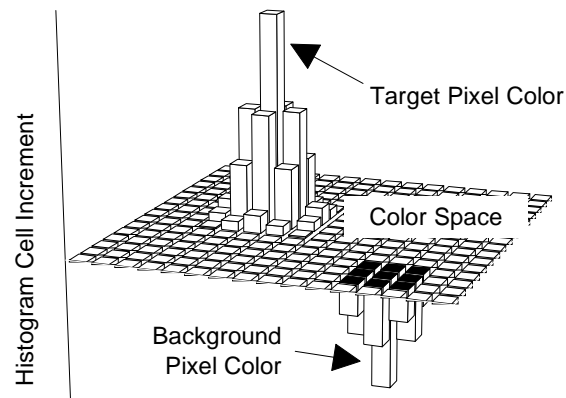


Figure 4: Weighting function around CP training data.

In order to reduce the effects of lighting, the segmentation should rely as much as possible on hue and saturation, ignoring intensity. However since the apparent color does shift somewhat with intensity we have compromised by quantizing the hue and saturation axes of the CP much finer than we do the intensity axis. This helps generalize across variation in lighting.

Training images are obtained by showing the user the live camera image with an outline of a hand gesture drawn over it. The user places their hand within the template and taps a key to freeze the image (Figure 5). While this produces imperfect training data, generally due to a slight mis-alignment of the



Figure 5: User training system.

hand and the template, the trained CP still produces a good segmentation for the reasons outlined in the design discussion below.

3.2.3 Segmentation process

The trained Color Predicate is used to segment skin from an image by backprojecting it onto the image in order to identify potential skin pixels. That is, the HSI value of each pixel is used to index a cell in the CP, and the value of that cell is used to label the pixel. This produces an image like that of Figure 3a where skin has a much higher density of labeled pixels than other regions. The dense regions can be identified by any one of several different techniques, and used as a mask to extract the hand from the original image. The details of the segmentation process depend on the purpose for which the segmentation is being used.

When tracking the motion of the hand, where segmentation quality is less important than speed, a reduced resolution image is undersampled from the original, the CP is backprojected onto it, and the single largest connected blob of pixels is identified (Figure 6a). This gives a good estimate of where the hand is in the image, but the details of the hand are obscured by the low resolution and segmentation errors such as missing regions or holes in the mask.

Pose recognition requires a more accurate segmentation. Here a high resolution window is cropped out of the input image (based on the location of the hand as determined by the tracking step), and the CP is backprojected onto it. This sub-image is then processed with a set of morphological operations designed to eliminate small regions, connect nearby large regions, and clean up larger regions by filling concavities and voids and blunting sharp convexities without changing the overall size of the region (for details see Section 3.7.2). Finally, the colors of pixels in the resulting region are

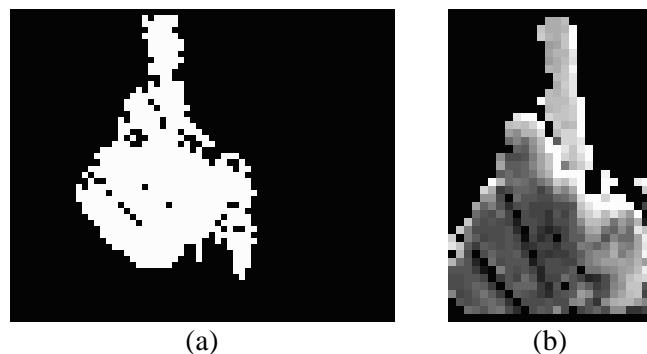


Figure 6: Segmentations of Figure 2 for (a) tracking and (b) pose recognition.

restored from the original image. The result is an image where the hand fills most of the image and the majority of the hand is present. Figure 6b shows an example that has been subsequently undersampled in order to be passed to the pose recognition network as described in Section 3.5.

3.2.4 Design Discussion

If it were possible to get perfect training data, i.e. every pixel inside the training template were skin and all possible skin colors were represented, then a straightforward histogram and threshold of the pixels in the training region would suffice to create a good quality Color Predicate. Unfortunately the training data is seldom perfect, and experience indicates that just histogramming the positive training examples does not produce a good segmentation. Noise in the color values and background pixels inadvertently inside the training region create false positive regions in the CP. This problem is aggravated because some target colors always seem to be poorly represented in the training data, creating false negative regions in the CP as well. Attempting to fill in these holes by using a large number of training images and/or a low threshold produces even more false positive regions. The modifications described above — i.e. using pixels both inside and outside the training template, and Gaussian weighting the histogram updates — improved the quality of the resulting segmentation dramatically. Analysis of the method shows why these adjustments are necessary.

Ignoring, for the moment, pixels with incorrect color values due to noise, and colors that occur both on the object to be segmented and in the background, there are four types of pixels in the training data. Pixels inside the training region with a target color (P_{IT}), pixels inside the training region with a color from the background (P_{IB}), pixels outside the training region with the target color (P_{OT}) and pixels outside the training region with the background color (P_{OB}) (Figure 7).

The optimal color predicate, S , is a thresholded histogram of $P_T = P_{IT} + P_{OT}$, which is not readily available. Further, since $P_I = P_{IT} + P_{IB}$, if we simply histogram P_I (all pixels inside the training region) we will end up with extraneous regions in the Color Predicate, leading to added noise in the segmentation (S' in figure 7).

The goal is to compute $P_T = P_I - P_{IB} + P_{OT}$, but without accurate knowledge of either P_{IB} or P_{OT} . Since the source of P_{IB} is a misalignment of the hand with the template during training, its members are pixels imaging background objects. Assuming that background

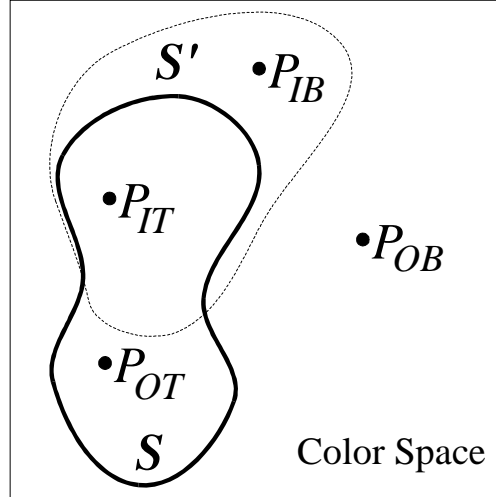


Figure 7: Optimal CP (S) and CP produced by histogramming the positive training examples (S').

objects are reasonably large and constant colored, then for every element of P_{IB} there are likely to be one or more elements of P_{OB} which are the same color (or more accurately, which have a color in the same histogram bucket). Indeed, since the misalignment of the hand with the training template generally results from a translation or scaling, not a topological change, the pixels in P_{IB} are likely to be at the edge of the template. In this case the pixels of P_{OB} lying on the other side of the template boundary are likely to image the same background object in about the same area, and so have approximately the same color.

Let

$$H_{IT} = \text{histogram}(P_{IT}),$$

and likewise for H_{IB} , H_{OT} and H_{OB} .

Then

$$\text{histogram}(P_I) = H_I = (H_{IT} + H_{IB})$$

and

$$H_O = H_{OT} + H_{OB}.$$

Now

$$\begin{aligned} H_I - H_O &= (H_{IT} + H_{IB}) - (H_{OT} + H_{OB}) \\ &= (H_{IT} - H_{OT}) + (H_{IB} - H_{OB}) \end{aligned}$$

Given the imaging conditions, the hand looms large in the image while the face, if present, is much smaller in the background. So assuming a reasonably accurate fit of the user's hand to the template, it follows that $|P_{IT}| \gg |P_{OT}|$. This implies that

$$(H_{IT} - H_{OT}) \cong H_{IT}$$

Again assuming the fit is reasonably accurate, there will be far more of the background outside the template than inside, in other words $|P_{OB}| \gg |P_{IB}|$. This implies that cells of $(H_{IB} - H_{OB})$ will have largely negative values.

Therefore,

$$\text{threshold}(H_I - H_O) \cong \text{threshold}(H_{IT})$$

Since $|P_{IT}| \gg |P_{OT}|$, it is safe to assume that all cells in H_{OT} with a positive count also have a positive count in H_{IT} , so that

$$\text{threshold}(H_{IT}) \cong \text{threshold}(H_{IT} + H_{OT}) = S$$

Using a small positive threshold cleans up further by removing the cells of H_{IB} which were not completely decremented by elements of P_{OB} . There will inevitably be a few cells of H_{IT} which were so decremented by H_{OT} that they will be eliminated from the desired region by the threshold, but their number seems to be relatively small.

The CP training algorithm described in Section 3.2.2 essentially computes $\text{threshold}(H_I - H_O)$ with the addition of a Gaussian smearing around each pixel's actual HSI value. The smearing reduces the reliance on the assumption that the colors of P_{IB} and P_{OB} match exactly. It also helps compensate for colors in P_{IT} which are poorly represented since neighboring cells help increase their support, so helping prevent voids inside S .

Another benefit of this training algorithm is that the relative amplitude of the Gaussian weighting functions for addition and subtraction can be used to adjust the behavior of the resulting CP when the background contains colors very similar to that of the target. When there are similar or identical colors in P_{IT} and P_{OB} , the boundary of S can not be determined precisely. Weighting addition more than subtraction forces the boundary toward the outside of S (expanding the region) and so including borderline colors during segmentation. Weighting subtraction more heavily shrinks S , excluding borderline

colors. So in effect the relative weight is a bias that forces the inevitable error to be segmented or not. In a “difficult” environment it should be set so as to exclude the questionable colors at the expense of losing some of the target object during segmentation. In an “easy” environment it can be set so as to include questionable colors, as there won't be many areas in the background of that color and more of the target will be picked up by the segmentation.

Figure 8 shows two-color predicates after training, but before thresholding. 8a was trained using a simple color histogram of the positive training data, 8b was trained using the full training algorithm. Figure 9 shows an image segmented with each of these maps. Notice that using the CP of 9a for segmentation, not only is there more background noise (false positive regions in the CP), but that there are more holes in the skin regions as well.

As will be shown in Chapter 4, this relatively simple segmentation algorithm achieves very good results. This is due both to the novel training algorithm and to environmental factors that have been exploited. Here we try to itemize some of those environmental

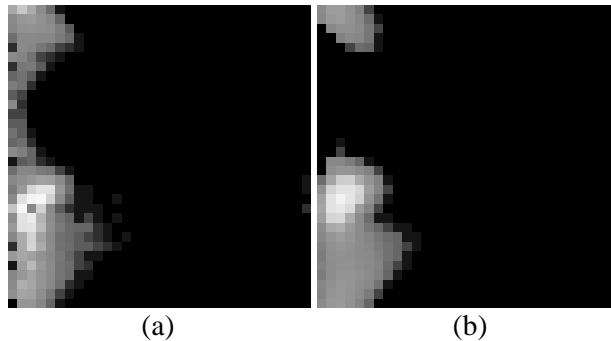


Figure 8: Color Predicates trained using simple update (a) and with Gaussian smoothing and subtraction (b). Values ≤ 0 are shown as black.



Figure 9: Images segmented using the CPs in Figure 8.

factors.

The fundamental assumption underlying the method is that skin coloration is relatively unique in the target environment. While there are often skin tone objects in an office, such as wood grain doors and desks, we rarely paint our walls or ceilings with skin-tone paints. This may be in part because we naturally want important objects in the environment to stand out rather than be camouflaged, and people are obviously important objects. Similarly, people rarely wear skin-tone clothes.

Another factor that contributes to the method's success is that in order to reduce eye strain offices are generally designed with strong uniform lighting, including light walls to bounce the light around and reduce shadows. Designing the lighting so people have good visibility, has the side benefit that vision systems generally have good lighting conditions.

The imaging geometry and typical position of a person using a computer assures that the hand looms large in the image, making it easy to find, even with imperfect techniques. Other skin blobs, including the user's face and other people in the background are guaranteed to be much smaller.

Finally, the properties of skin itself make the task easier. Skin is primarily Lambertian and somewhat translucent. Therefore from most viewing angles, light received from it has little specular component and has been filtered to contain primarily skin tones. This reduces color variations due to light source coloration, inter-reflections, surface normal direction, and the like.

This segmentation algorithm was inspired by Swain's work using color histogram matching for object identification. In [Sw90] he trained an RGB histogram using images of the desired object. He was then able to identify images containing that same object by matching the trained histogram against the histogram of the new image. The approach worked best on objects containing many strongly colored regions.

Others have developed more elaborate algorithms for color segmentation. Buluswar and Draper have developed a color based segmentation scheme for identifying target candidates in outdoor scenes [BD94]. They hypothesize that reflections from colored objects under daylight will always fall on a smooth surface in RGB space regardless of the specific lighting conditions. They train multivariate decision trees to identify a piecewise linear approximation to that surface, then use them to classify pixels and segment target objects in a scene. The chief advantage of this approach is the generalization it provides. The shape of the regions produced in color space were matched to the observed shapes produced by objects under outdoor illumination, so the

approach is predisposed to produce solid regions with the correct characteristics. The disadvantage is the relatively long training times required, which works against having a user train the system interactively. The approach outlined above produces solid well-formed regions in color space using a simpler approach.

3.3 Hand Tracking

Determining the position of the hand is the next essential step for interpretation of the gesture. From the position we can determine if the user is gesticulating and where they are pointing on the screen. The sequence of positions form the input for the motion feature extraction routines, which are central to interpreting the gesture. Thus fast and accurate determination of position is essential.

From the segmented hand we can easily determine where the user's hand is within the image. The user, however, is not thinking in terms of the location of their hand in the image, rather with respect to the screen (this is obvious for pointing gestures, but also seems to apply to all gesticulation. Since the screen image is what is being manipulated, it is natural to think in that coordinate system). To best interpret the user's actions, the system must translate from the image location of the hand to screen coordinates before attempting to interpret the motion.

When the user points to point (x_s, y_s) on the screen, his hand creates a blob with centroid (x_I, y_I) . If that mapping were relatively independent of the hand's location in the image, an affine transformation between image-space and screen space would suffice. However the mapping of (x_I, y_I) to (x_s, y_s) changes as the user points to different screen locations primarily due to three causes: the apparent shape of the hand changes dramatically as its orientation with respect to the camera changes, the imaging geometry creates strong perspective, and factors such as segmentation noise vary consistently in different locations. Figure 10 shows an example of how the segmented hand images change as the user points to the four corners of the screen. Note that the extended finger has been cropped off by the top of the image when the user points to the top of the screen. Note also that when the user points to the bottom left of the screen the face is segmented along with the hand as well as the finger being cropped off.

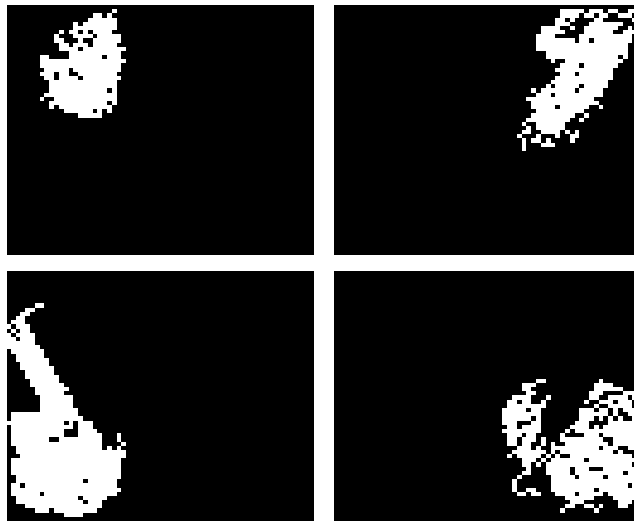


Figure 10: Tracking resolution segmented images of the user pointing to the four corners of the screen (left and right are reversed for clarity, that is, the images on the left are actually the user pointing to the right edge of the screen).

Assuming that the shape of the hand blob in one corner of the screen morphs uniformly into the shape of the hand blob in an adjacent corner, and ignoring segmentation noise, the relative displacement of the desired cursor location with respect to the centroid of the hand blob will vary monotonically across the screen. This implies that the centroids of the hand blob form a quadrilateral in image space as the user traces the outline of the screen (Figure 11). Thus, the mapping between image-space and screen-space can be approximated with an inverse perspective transformation between the coordinate systems, which maps an arbitrary quadrilateral to a rectangle.

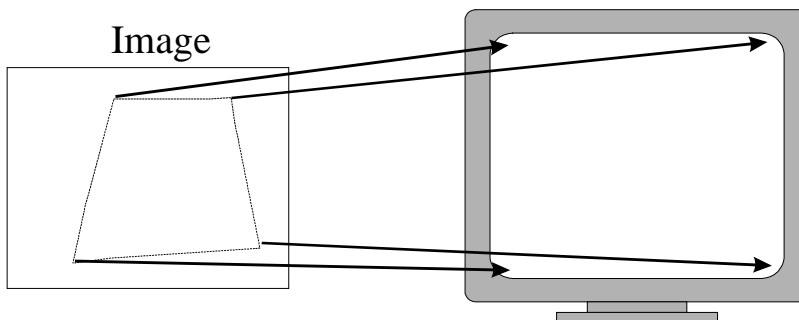


Figure 11: The centroid of the user's hand pointing to the corners of the screen forms a quadrilateral in image space.

This transformation has the form

$$[x', y', w'] = [x_I, y_I, w_I]A$$

where

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

is the inverse perspective transformation matrix. Then

$$x_S = \frac{x'}{w'} = \frac{A_{11}x_I + A_{21}y_I + A_{31}}{A_{13}x_I + A_{23}y_I + A_{33}}$$

and

$$y_S = \frac{y'}{w'} = \frac{A_{12}x_I + A_{22}y_I + A_{32}}{A_{13}x_I + A_{23}y_I + A_{33}}$$

After applying this transformation to the screen location, (x_S, y_S) , it is combined with the hand blob's size S , and passed to the host as an XYS token for use by the rest of the system.

To infer the transform coefficients A_{ij} , the user points to the four corners of the screen. Considering the screen, for the moment, as a unit square gives the four $(x_I, y_I) \leftrightarrow (x_S, y_S)$ pairs:

$$(x_0, y_0) \leftrightarrow (0, 0)$$

$$(x_1, y_1) \leftrightarrow (1, 0)$$

$$(x_2, y_2) \leftrightarrow (1, 1)$$

$$(x_3, y_3) \leftrightarrow (0, 1)$$

The easiest way to derive the transform coefficients is to first derive the coefficients for its inverse, that is the transformation from screen coordinates to image coordinates, then invert the transformation matrix. Call this inverse transform

$$A^{-1} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

For this mapping the transformation equations become:

$$x_I = \frac{a_{11}x_S + a_{21}y_S + a_{31}}{a_{13}x_S + a_{23}y_S + a_{33}}$$

and

$$y_I = \frac{a_{12}x_S + a_{22}y_S + a_{32}}{a_{13}x_S + a_{23}y_S + a_{33}}$$

Combining these with the $(x_I, y_I) \leftrightarrow (x_S, y_S)$ pairs gives:

$$a_{11} = x_1 - x_0 + a_{13}x_1$$

$$a_{21} = x_3 - x_0 + a_{23}x_3$$

$$a_{31} = x_0$$

$$a_{12} = y_1 - y_0 + a_{13}y_1$$

$$a_{22} = y_3 - y_0 + a_{23}y_3$$

$$a_{32} = y_0$$

$$a_{13} = \frac{\begin{vmatrix} \Delta x_3 & \Delta x_2 \\ \Delta y_3 & \Delta y_2 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}}$$

$$a_{23} = \frac{\begin{vmatrix} \Delta x_1 & \Delta x_3 \\ \Delta y_1 & \Delta y_3 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}}$$

$$a_{33} = 1$$

where

$$\Delta x_1 = x_1 - x_2$$

$$\Delta x_2 = x_3 - x_2$$

$$\Delta x_3 = x_0 - x_1 + x_2 - x_3$$

$$\Delta y_1 = y_1 - y_2$$

$$\Delta y_2 = y_3 - y_2$$

$$\Delta y_3 = y_0 - y_1 + y_2 - y_3$$

Since, for a perspective transformation the adjoint of the transformation matrix is the equivalent of the inverse [Wo94]:

$$A' = \begin{bmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix}$$

where A' is the mapping from image coordinates to the unit square. This is then multiplied by the transformation matrix, T , that maps the unit square into the actual screen coordinates

$$A = A'T$$

giving the full transformation matrix from image coordinates to screen coordinates, A .

3.3.1 Design Discussion

Although the perspective transformation between image and screen coordinates is adequate for the prototype, it does not handle all the types of distortion present in the domain. First the assumption that the hand outline varies uniformly across the screen is just an approximation. There are local variations as, for example, the extended finger passes close to the camera when the user points to the bottom center of the screen. There is also segmentation noise that varies predictably with the location of the hand in the image. For example, in some regions of the image one side of the hand may tend to break away from the bulk of the hand due to a specular reflection. Thus the area in image space where the hand centroid maps to a rectangular grid on the screen does not actually form a quadrilateral, but a warped grid, whose corners form a quadrilateral (Figure 12). As a result, the mapping can not be accurately captured by a simple perspective warp.

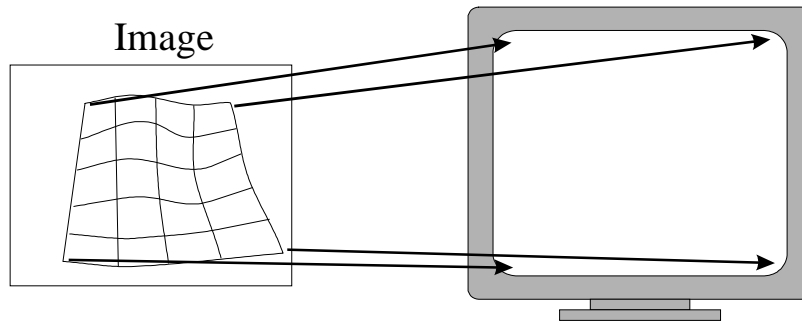


Figure 12: The centroid of the hand as it follows a grid in screen space, forms a warped grid in image space.

This complex mapping can be better represented by a second degree polynomial transformation. The data needed to derive the polynomial coefficients can be obtained by having the user track a dot on the screen as it is moved back and forth along a grid covering all areas of the screen. The dense set of point correspondences can then be used to compute the polynomial coefficients using a weighted least squares. Refer to [Wo94] for details. We expect a polynomial warping will provide a noticeable improvement in how well the cursor can track the hand with little additional runtime overhead. The training procedure should not be excessively burdensome for the user, but the time needed to derive the tracking coefficients after training will likely increase.

Any static image-to-screen mapping relies on the assumption that the relative position of the hand centroid and the desired screen location does not change over time. We can safely assume that the relationship of the camera to the screen will not change, but it is more of a stretch to assume that the user will be consistent in how they gesticulate. The assumption may not hold as, for example, the user changes their seating posture or begins to hold their hand differently as they tire. This problem can be most acute with new users, who take some time to settle into a consistent pointing pattern.

There is no clear solution for this problem. Some amount of variation is acceptable in the mapping as the user will compensate using visual feedback. Our experience with this system indicates that the variations caused by minor shifts in position do not significantly affect the mapping, and are easily handled by the user feedback loop. Significant changes in user posture, such as the user pointing to the screen while standing or the changes in pointing pose during the first few minutes someone uses the system, cause large amounts of positional error. This problem has only been addressed here by providing a calibration procedure that is quick and easy to use so that retraining is relatively painless.

3.4 Motion

Given a stream of tokens representing the position and size of the hand in screen coordinates, the next step is to place the cursor on the screen to provide the user with feedback, and to extract the salient features of the user's hand movements. If the raw XYS data is used without any smoothing, the cursor jumps around in a very annoying fashion and the motion features are difficult to extract (for an analysis of the types of noise in the raw XYS data, see the design discussion at the end of this section). Therefore the XYS token stream is smoothed using an algorithm designed to compensate for the noise and restore, as much as possible, a cursor movement that is pleasing to the user and from which the motion features can be reliably extracted. This section describes the smoothing algorithm and the motion feature extraction routines.

3.4.1 Smoothing the Hand Path

In order for the user to be able to comfortably position the cursor, there are several criteria that we have empirically determined. The cursor must remain relatively still when the hand is not moving. Some slow wandering is acceptable, but it must not jump around. Somewhat at odds with this, the cursor must respond to small movements almost immediately so that fine positioning is possible. It is also important that the cursor not lag far behind when the hand is moving quickly so that the user can point to an object and retract their hand in a smooth motion, and rely on the cursor to have detected the endpoint of that motion. The cursor must not, however, overshoot or oscillate when the motion of the hand suddenly stops.

Since the cursor is tracking a physical object, namely the user's hand/arm system that has the properties of mass and inertia, we began by giving it similar properties. In particular the cursor is treated as having mass (M) and velocity (V_i) as well as position (X_i). Each time step (i), the difference between the current cursor location (X_{i-1}) and the current XYS token (H_i), is converted to a force that accelerates the cursor in the appropriate direction. This basic inertial model meets several of the above requirements. The inertia of the cursor damps the effect of high frequency noise, but allows it to immediately begin moving in the direction of an applied force. If a force continues in the same direction, the cursor quickly accelerates in that direction to follow. In order to improve both the damping of the cursor when the hand is still and the speed of tracking large movements, the distance between H_i and X_i is converted to a force using a non-

linear “spring” function.

We now describe the algorithm in detail. Note that this description is based on the one-dimensional case, so that vectors have only a sign and a magnitude. Since the algorithm is applied at discrete time intervals, all velocities and accelerations are with respect to that time-step, so that a velocity of x implies a motion of x pixels in one time-step. The generalization to two dimensions will be discussed later in this section.

Given a displacement from the current cursor location to the hand location:

$$D_i = H_i - X_{i-1}$$

we want to compute some force, $F_i = f(D_i)$, and apply it to the cursor using

$$A_i = F_i / M, \quad V_i = V_{i-1} + A_i \quad \text{and} \quad X_i = X_{i-1} + V_i$$

such that X_i tracks the hand as described above.

A simple linear spring model does not give good results. A spring constant low enough (or, equivalently a mass high enough) to damp the noise when the hand is still does not give fast enough response to larger hand movements. Thus the mapping function, f , must be adjusted.

Consider the situation where the cursor is essentially motionless at the beginning of the cycle (i.e. $V_{i-1} \cong 0$). A small displacement should cause the cursor to accelerate toward the hand, but since there is a reasonable chance that the displacement is made up primarily of noise, the acceleration should be modest. Let F_{\min} be the minimum force that should be applied, its value, for the moment, unspecified.

To avoid overshooting the hand, the maximum force applied to the cursor should accelerate it toward the hand just fast enough to catch up in this cycle. Setting

$$F_{\max} = M(D_i - V_{i-1}) \quad (\text{Equation 1})$$

leaves, at the end of this cycle, $X_i = H_i$.

Since the magnitude of any noise is independent of the size of D (see the noise model in Section 3.4.3), as D_i becomes larger it is more likely that it represents approximately the actual displacement. The mapping from D_i to F_i , then, should use a function that will approximate F_{\min} on small displacements, and approximate F_{\max} on large displacements.

A good candidate is the sigmoid function:

$$F_i = \frac{F_{\max} - F_{\min}}{1 + e^{-\left(\frac{D_i - k}{\mu}\right)}} + F_{\min} \quad (\text{Equation 2})$$

A plot of Equation 2 is shown in Figure 13. The parameters k and μ can be set to give the desired response given the noise characteristics in the domain. k determines the knee of the sigmoid and μ determines its slope. In general, small displacements approximate a low spring constant, allowing the cursor to be pulled toward the hand, but gently. Large displacements approximate a high spring constant to get it moving quickly. The knee of the function (k) can be adjusted so that it occurs at a higher D than most of the random jitter in H that occurs when the hand is still, helping to damp out low amplitude noise.

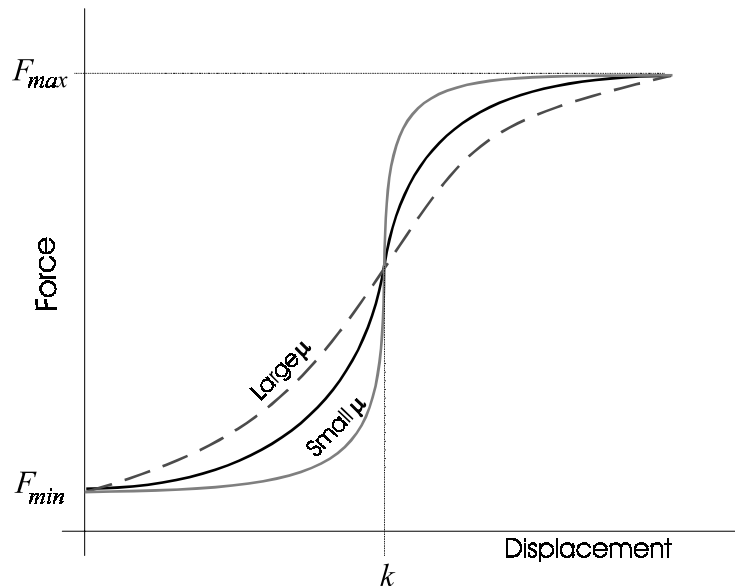


Figure 13: Sigmoid force scaling functions.

While this forms the basis of the smoothing model, more generally, the cursor is moving at the beginning of a cycle. Therefore there are three possible scenarios (in all cases the direction of the current cursor movement is taken as positive):

Case 1: $D_i > V_{i-1}$, so that at the current velocity the cursor will not catch up with the hand in one cycle.

Case 2: $0 \leq D_i \leq V_{i-1}$, so that at the current velocity, the cursor will catch up to or pass the hand in this cycle.

Case 3: $D_i < 0$. Here the velocity of the cursor is carrying it away from the hand.

In all three cases, the cursor should accelerate no more than it takes to reach the hand in one cycle, so that Equation 1 still holds. However note that the term F_{\max} is now something of a misnomer, as in Cases 2 and 3 it actually represents the most negative force that should be applied.

First, consider Case 1. Here, at a minimum, the cursor should continue on toward the hand at its current velocity, thus

$$F_{\min} = 0$$

At larger displacements the cursor should accelerate further toward the hand, so F_i is computed as per Equation 2.

In Case 2 the cursor should decelerate so as not to pass the hand this cycle, however since it is initially traveling fast enough to catch the hand this cycle, it should decelerate no more than that. Thus we have that

$$F_i = F_{\max}$$

Finally, consider Case 3. For this situation to occur, either the hand must have changed direction since the last cycle, or it must be moving slowly enough that noise in the perceived location makes it appear behind the cursor at the beginning of the cycle. If the cursor were allowed to retain its initial velocity, a simple inertial model would give the situation illustrated in Figure 14, where the cursor flies off away from the hand for some period of time as it decelerates and returns, causing overshoot and oscillation. Thus at a minimum the cursor must stop, so that

$$F_{\min} = -M \times V_{i-1}.$$

The cursor should then chase the hand with the same behavior described above, but of

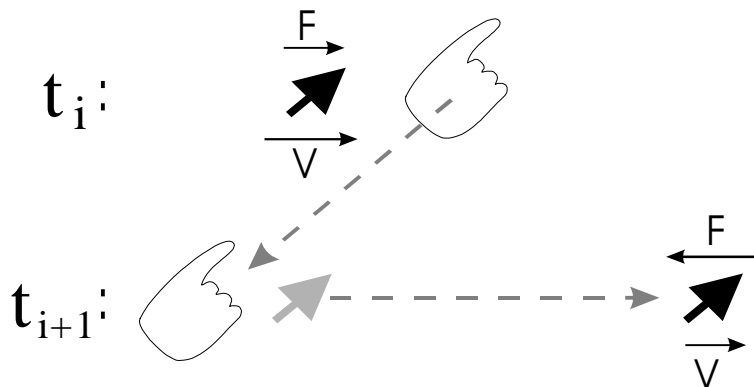


Figure 14: The hand “backing up” behind the cursor between cycles and causing overshoot in a simple smoothing algorithm.

course inverted in direction. Thus

$$F_i = \frac{F_{\max} - F_{\min}}{1 + e^{-\left(\frac{-D_i - k}{\mu}\right)}} + F_{\min}$$

Note the negation of D_i .

A good way to envision the combined effect of these cases is to plot the force applied to the cursor (which is proportional to its acceleration) versus D_i . Figure 15 shows force curve for various values of initial velocity. For reference, F_{\max} is also plotted (recall that at F_{\max} the cursor will track the perceived hand location exactly). Notice that in Case 3, the behavior is independent of initial velocity. The cursor always is given enough force to stop, then heads toward the hand with an acceleration that depends on D_i . In Case 2 the cursor always decelerates at F_{\max} so that it meets the hand this cycle. The behavior in Case 1 depends strongly on initial velocity. When V_{i-1} is small, the cursor chases the hand as per the sigmoid function. At high initial velocity it follows the hand at F_{\max} as in Case 2. Intermediate velocities produce behavior between the two.

The net effect is that the cursor always moves toward the hand, but never passes it. When the hand has been moving slowly, additional small movements cause the cursor to gently accelerate toward it, while large movements cause the cursor to fly after it and catch up quickly. Behavior changes smoothly between these extremes. When the hand has been moving rapidly, the cursor tracks any displacement closely. This fits the requirements for cursor behavior laid out at the beginning of this section, and produces a cursor motion which feels natural, and is easy for the user to control (see Section 4.2 for a more detailed evaluation).

Currently this algorithm is applied to each dimension of the motion (X, Y and S) independently. Due to the non-linear nature of the force scaling function, this is not identical to applying a similar algorithm in three dimensions, but nonetheless it produces good results. We have explored various ways to generalize this algorithm to more dimensions. The most promising is to project V_{i-1} onto D_i , then continue as in the one-dimensional case.

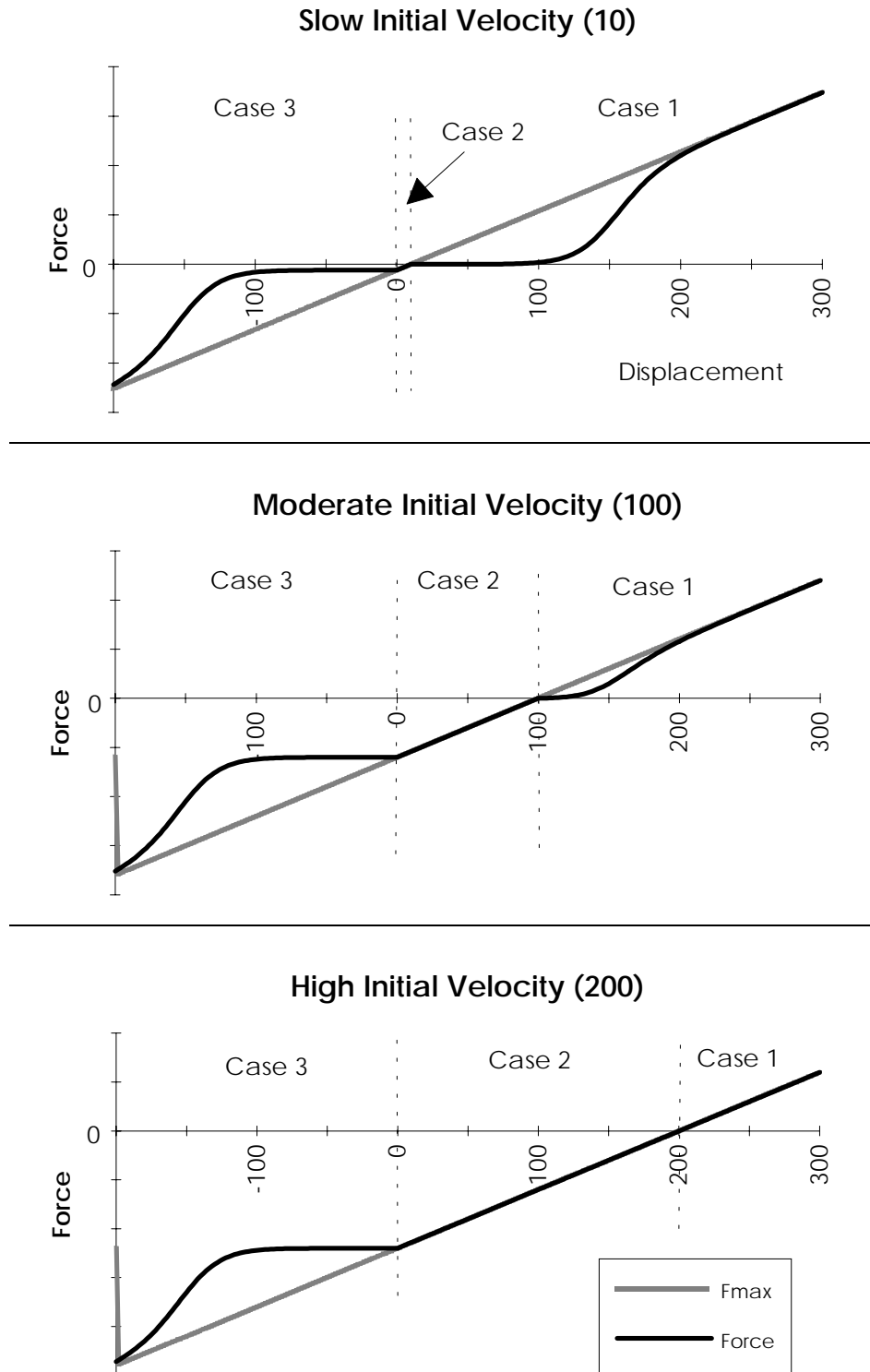


Figure 15: Force applied to the cursor versus hand displacement for three initial velocities. A force of F_{max} gets the cursor exactly to the current hand position by the end of the current cycle.

3.4.2 Extracting Motion Features

As each XYS token arrives, it is smoothed using the algorithm described above and placed in a short circular buffer. When the interpretation of the interaction reaches a state in the transition network where one possible link is a motion feature, this buffer is examined for the presence of that feature. For example, at the state labeled IgnoreMvt1 in the interaction language shown in the Appendix, the buffer will be examined for evidence of the hand moving up (an decrease in Y by some threshold) or that the hand is gone (S falling below some size threshold).

The current version of the system supports the 17 motion features shown in Table 1. All are Booleans that are true if the feature template matches the current token history. Identification of the majority of these is straightforward. Since smoothing has already been done, the token stream is examined for only the last two cycles to determine features such as up, down, left, right, forward and back. These features are conditional on HandPresent and not(Pause). There are thresholds for slow and fast motion. The hand is considered paused if the total movement over two cycles is below a threshold. Detection

Base Feature	Variations	Description
Up	Fast/Slow	Toward top of screen
Down	Fast/Slow	Toward bottom of screen
Left	Fast/Slow	Toward user's left of screen
Right	Fast/Slow	Toward user's right of screen
Forward		Moving toward screen
Back		Moving away from screen
Pause		Lack of movement
Reverse	X/Y	Reversal of direction of movement
Comma		Short Back/Down - Forward/Up movement
HandPresent		Hand size above threshold
HandAppear		HandPresent just became TRUE
HandGone		HandPresent just became FALSE

Table 1: Motion Features.

of reversal is straightforward. HandGone and HandAppear are only flagged when HandPresent changes between TRUE and FALSE.

Comma is a unique feature that was discovered as we used the first working versions of the system. A Comma is defined as the user moving their hand smoothly back away from the screen then toward it again. The movement can be quite small, as when the pointing hand is tilted back at the wrist and then forward again. It is very useful in situations where the user wishes to end one phrase of the gestural sentence and begin another — for example to select the window they are pointing to and point at it again to move it. A Comma is detected when two cycles previously the hand was moving down and away from the screen, and the current cycle the hand is moving up and toward it, and the change in velocity of the hand is greater than a threshold (V_c). More specifically $Comma_i$ is a Boolean value such that

$$Comma_i = Down_{i-2} \& Back_{i-2} \& Up_i \& Forward_i \& (|V_{i-2} - V_i| > V_c)$$

Note that the camera perspective, looking up as well as toward the user, means a backward movement of the hand is detected not only as a decrease in size, but a downward movement in Y, so that the user only needs to move their hand away from and then toward the screen, not explicitly down and up, to trigger a Comma. The threshold on change in velocity is needed to prevent small amounts of jitter in the motion from being detected as a Comma.

3.4.3 Design Discussion

The noise in the raw XY token stream has very distinctive characteristics. This section will discuss these characteristics in some detail, and examine their effects on the system.

First the noise can be subdivided into a stable "DC" component — to use an electrical analogy — and a time-varying "AC" component. The DC component consists of an offset from the "ideal" cursor location. If the user is pointing to a screen location, for example, the cursor might appear somewhat off to the side. It is caused by local segmentation errors which are not well compensated for by the image-to-screen mapping algorithm of Section 3.3, and by variations in hand shape. Its value varies, depending on the location and pose of the hand, but when the hand is still, it remains constant.

Of more interest to the smoothing algorithm are the AC components of the noise.

These only show up when a sequence of images are considered. The AC noise can be further subdivided into a “constant” component that is always present whether the hand is moving or not, and a “variable” component that shows up only when the hand is moving. The constant AC noise is caused by random errors in the segmentation and small unintentional movements of the hand. While this component is always present, amplitude and frequency vary with screen location. The variable AC component is caused by changes in the DC noise component as the hand moves.

In order to understand the effect of this noise on the interpretation of the hand's path it is more interesting to view the AC components of the noise grouped another way.

Type 1 - High frequency, low amplitude jitter

This results from pixels whose color is on the borders of regions in the color predicate. Small amounts of image noise between frames cause them to be segmented or not in subsequent images. Because the odds of a particular pixel being segmented as part of the blob or not are roughly equal, the effect of some pixels being added in and others being omitted balances out, so the amplitude of the jitter is never very great. The result is a frame-to-frame variation of a few pixels in X, Y and S. This is the principal component of the constant AC noise described above, and occurs whether the user is moving his hand or not. Unsmoothed, this noise can make it very difficult to reliably extract motion features. It is also very annoying to the user, especially when he is trying to precisely position the cursor.

Type 2 - Low frequency step or ramp functions

As the hand moves through a border between regions where the DC noise component has different characteristics, the XYS token stream undergoes a step or ramp. The amplitude of the step depends on the difference between the DC offsets and can range from small to moderately large. The shape of the step depends on the nature of the border (region borders can be several pixels wide), the speed of the hand, and the relative orientation of the border and the path of the hand. This noise obviously only occurs when the hand is moving. In this domain, where the exact path of the hand is not important to the user, it is not very distracting to them. It will be more of a problem in applications where the path is important, such as drawing with the fingertip. This type of noise can occasionally cause problems with the motion feature extraction routines, as will be described below.

Type 3 - Medium frequency, medium amplitude variations

This occurs when the hand is on the boundary of two DC regions, or is in an area of the image where the segmentation is unstable for some other reason (e.g., a finger may be connected to the hand by a thin thread so random noise causes it to break away and reconnect from the bulk of the hand). Either way the segmentation jumps back and forth between significantly different interpretations. This noise can be present whether the hand is moving or still. The amplitude can range from insignificant to 100 or more screen pixels, implying 1 to 2 inches of screen distance. Type 3 noise is extremely distracting for the user and can cause problems with the motion feature extraction operators.

Type 4 - Sporadic high amplitude jumps

Occasionally the token stream will jump for a cycle or two to a distant location, and return. This is caused by poor imaging during fast motion, which in turn causes large pieces of the segmented hand blob to break away or the blob to disintegrate into enough small pieces that the hand is no longer the largest connected component. Since it generally occurs during fast motion it is not terribly distracting to the user, but it can be very destructive to the motion feature extraction operators.

Type 1 noise is well handled by the cursor motion smoothing algorithm. The inertia of the cursor helps it resist small random displacements. The sigmoid scaling of F also helps damp it by scaling down the force generated by displacements of this amplitude. The wandering that remains after smoothing is of low enough frequency that the user can position the cursor easily, although, as will be shown in Section 4.2.2, it does increase the time and effort it takes to interact with small objects. The feature extraction operators are not significantly effected, as the noise is easily ignored by a low threshold.

Type 4 noise is also very well damped by the smoothing algorithm. Because of its short duration the cursor will only move slightly in the direction of the jump before the hand “returns” to its old path, pulling the cursor back.

Type 2 noise can be decomposed into two components, step functions in the direction of the hand's motion, and step functions perpendicular to it. The potential ill effects of the noise component parallel to the direction of travel are well handled by inertia and the checks in the smoothing model for overshoot. The component perpendicular to the direction of travel is smoothed only slightly. Fortunately this component is transparent to the user, since the hand is generally moving rapidly at the time. The noise it causes in the

motion feature extraction operators has not proven to be a problem. Some of this is likely due to the design of the interaction language. When the hand is moving from one place to another, all motion features are ignored until a Pause is detected. The perpendicular component of Type 2 noise are not likely to cause a Pause to be detected.

The most problematic is Type 3 noise. Under some conditions it passes through the smoothing algorithm with reduced but still significant amplitude because the original amplitude is high enough and the frequency low enough that the cursor begins to move toward it. Then, even in the smoothed path, the cursor can jump back and forth between screen locations an inch or more apart. This can be a significant problem for the user when they are trying to finely position the cursor. It also causes problems with motion feature detection, creating false positive features and inhibiting reliable detection of important features such as Pause.

Type 3 noise is not always present, and it is often very localized, but under some conditions it can cause a portion of the screen to be almost unusable. When this occurs, the problem can usually be significantly reduced by adjusting the cursor mass and the shape of the sigmoid weighting function with k and μ . This is not an ideal solution, however, because the range of parameters that damp out the noise without destroying good cursor tracking in general can be very narrow. A better solution is left for future work.

Many components of the noise spectrum can be reduced by increasing the resolution of the image used during tracking. Type 1 noise is smoothed by the statistical smoothing of a larger blob size. Type 2 and the problematic Type 3 noise will be reduced because large scale local segmentation errors will be reduced. For example if a finger has more pixels across its width, it will be more resistant to being lost completely when a few pixels on its edge are distorted by blooming around ceiling lights. In order to increase tracking resolution without degrading response time, the execution speed of the algorithms will have to be improved. This will be discussed in more detail in Section 5.1.2.

An often suggested alternative to this smoothing algorithm is a more physically realistic damped spring model, where there is a damping force dependent on velocity. This was not used, primarily because of the sigmoid scaling function on F . Empirically we have found that this type of non-linear response is essential to getting cursor movement that is pleasing to the user. In order to avoid overshoot and oscillation, and at the same time get fast response, a damped system would have to be tuned to critical

damping, but given the non-linear force scaling, determination of the damping parameters would have been very difficult.

3.5 Pose Recognition

During a gestural interaction, the system navigates through the transition network describing the interaction language, as will be described in Section 3.6. When a node is reached which branches based on the pose of the hand, a high resolution sub-image is cropped tightly around the hand using the current XY token. That sub-image is segmented as described above, then pre-processed and passed to a neural network that has been trained to differentiate the poses associated with the various branches from the current node. Based on the classification, traversal of the transition network continues. This section will now describe the classification process in more detail.

When a pose needs to be classified, the image containing it is first cropped and preprocessed such that the hand fills a known size image, and variations due to lighting are, as much as possible, removed. First the XY token is used to crop the hand out of the full resolution image. The location of the crop window is determined by X and Y. Y is used to select from a fixed range of cropping window sizes — smaller near the top of the screen, and larger near the bottom (see Section 3.5.1 for details). This results in full color image of around 200 by 300 pixels with the hand approximately centered.

Next the cropped image is undersampled to a fixed size (currently 26x39) using bilinear interpolation and segmented as described in Section 3.2. The color of the segmented hand region is converted to gray using an algorithm designed to highlight intensity features on the interior of the hand blob. Specifically $M = cI - R_n$ where M is the resulting gray value. $I = \frac{(R+G+B)}{3}$ is the intensity of a pixel. R_n is the normalized red channel, that is $R_n = r \times I_m$ where $r = \frac{R}{(R+G+B)}$ and I_m is the maximum intensity value, typically $2^n - 1$ where n is the number of bits per pixel. c is an empirically determined constant. It will be described further in the discussion of the color-to-gray algorithm in Section 3.5.1.

The resulting gray image is histogram equalized. This distributes gray values across the available dynamic range, and helps eliminate differences in overall intensity between images.

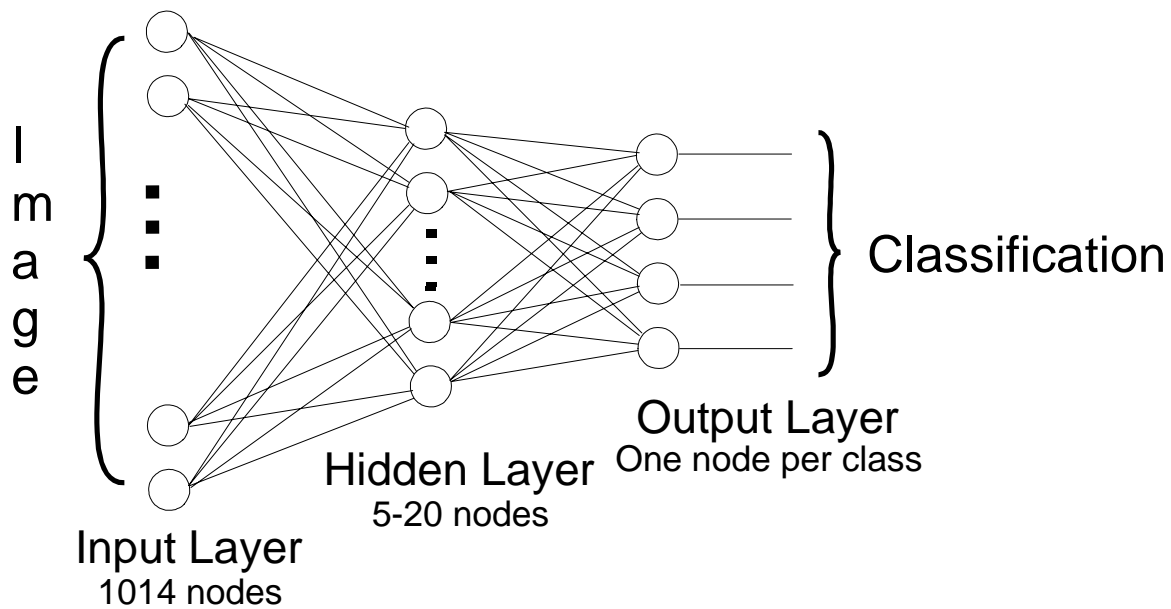


Figure 16: Pose recognition network architecture.

This image is then passed to the neural network as its input vector. Each node in the transition network which branches based on pose classification has a unique network trained to differentiate only those poses. Every net has 1014 input nodes (one for each pixel in the input image), 20 hidden nodes and one output node for each classification this network may make (Figure 16). After an image is presented and node activation propagated, the output node with the highest value is taken as the classification of the pose.

The neural net is trained by presenting it preclassified images of the poses it is to distinguish, and using traditional backpropagation to adjust the weights [Fu94]. The training images are randomly modified in translation, rotation and scale (details in Section 3.7.5) This helps the network generalize by keeping it from simply memorizing the training images, and effectively increases the size of the training set.

The network is trained in two stages. First it is trained on a set of some 40 images of each pose captured as described below and automatically segmented. The training images are presented in a different random order each training cycle. Training continues until performance stabilizes, typically at something over 90%. When this network is put to use controlling the system, however, an unacceptable number of misclassifications are still made. The network is therefore trained a second time, using images it misclassified after the first step. To obtain these images, the system is used for several minutes, and any pose images that have been misclassified are captured and added to the original

training set. The network is then trained on the new, larger set until performance stabilizes. The final network shows about the same accuracy on the pre-compiled test sets as it did before the final phase of training, but performance on live data is noticeably better. For more discussion of network training see Sections 3.5.1 and 4.3.2.

To obtain training images with a representative collection of views of each pose for the first stage of training, an image is taken with the user “aiming” the pose at each point on a grid superimposed on the screen. The location and density of the grid depend on the pose. For pointing poses, a 4x5 grid covers the whole screen, as any location on it is a potential target. For other command poses, the grid typically does not cover the top third of the screen as a user will seldom raise their hands to aim the pose that high without reason. The grid does also not typically cover the “opposite” edge of the screen from the user's handed-ness. That is, right handed users will seldom aim a command gesture at the left edge of the screen, as it is awkward to move their hand across their body to do so. Thus the grid for command poses will not cover the opposite quarter of the screen .

3.5.1 Design Discussion

The pose classification task is somewhat more difficult than it first appears. While there are relatively few poses to be differentiated, there is large variation in the appearance of the poses depending on where the hand is in the work space and exactly how the user holds the pose. This is particularly true of the pointing pose, which varies



Figure 17: Various appearances the Point pose takes on.

greatly depending on where on the screen the user is pointing (Figure 17). Other problems include the variation in scale of the hand within the cropped image and segmentation errors such as an extended finger being missed, holes in the hand blob, the hand being cropped by edge of the image or features such as the face being segmented along with the hand.

Image Preprocessing

The primary goal of the preprocessing algorithm is to ensure a uniform input to the classification networks. This must include scaling the hand to a reasonable size range, removing any background, and converting from color to gray. It is also desirable to reduce noise such as lighting variations and emphasize important details such as the shadows between fingers.

The size of the cropped image is based on the Y component of its location because, due to the imaging geometry, the apparent size of the hand gets larger when it is lower in the image (see Figure 1 and Figure 18). Since users tend to gesticulate at a relatively constant distance from the screen, this effect is quite reliable.

Contrary to expectations, using the Size component of the XYs token to determine cropping window size did not produce good results. The Size parameter is based on the number of pixels in the segmented hand blob. Due to different “densities” of various poses, the pixel count of the pose is not a good measure of bounding box size. For example a pose with all fingers wide spread will need a much larger bounding box than a pose where all the fingers are squeezed together, but the pixel counts could easily be the same. A pointing pose will have only slightly more pixels in it than a fist (from the extended finger) but its bounding box is significantly bigger. This is further complicated by the location dependent segmentation errors discussed in Section 3.4.3.

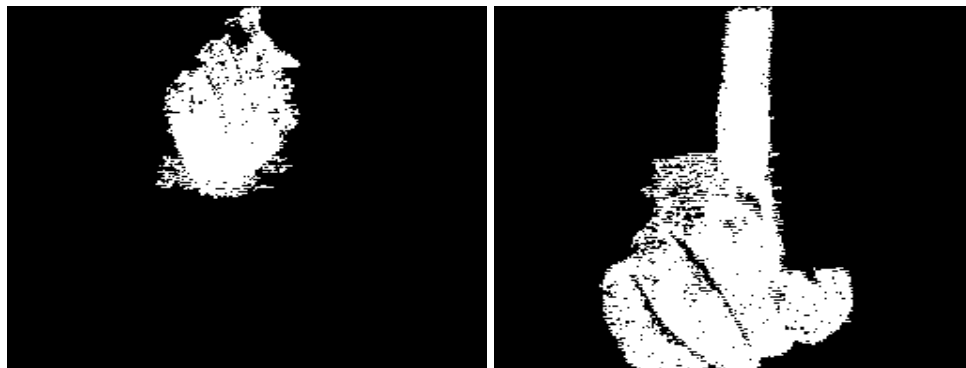


Figure 18: Hand pointing to the top and bottom of the screen.

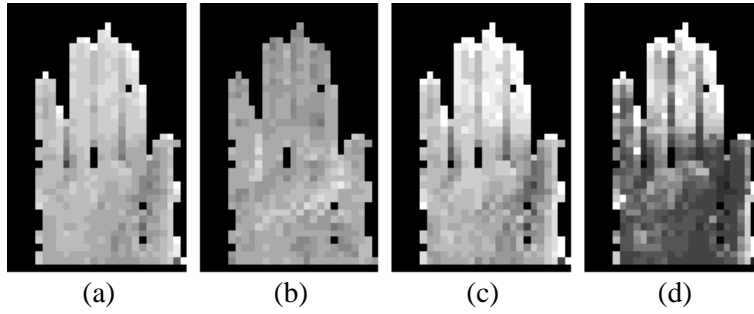


Figure 19: Color to gray conversion.

(a) Intensity. (b) Normalized red channel. (c) $3I - R_n$. (d) c plus histogram equalization.

For converting the color signal to gray, the obvious choice is to use the intensity of the color signal ($(R+G+B)/3$). Unfortunately, this results in a washed out image of the hand (Figure 19a) where details such as the limbs (receding edges) of fingers, local shape variation and skin creases are not well differentiated. This is likely due to a combination of diffuse lighting and skin being a highly Lambertian surface. Under these conditions the intensity of skin stays relatively insensitive to the orientation of the surface normal with respect to viewing direction.

The color-to-gray conversion algorithm, $M = cI - R_n$, was developed to bring out features within the boundary of the hand, and so help the classification networks base decisions on more than just the outline of the segmented region. The normalized red channel of the color signal ($R/(R+G+B)$) is actually brighter in shadows than where the hand is well lit (Figure 19b). Thus subtracting it from the intensity channel helps highlight features of the hand, including creases and the lines between fingers (Figure 19c).

The scale factor c is needed to balance the contributions of I and r . Its value depends on the imaging conditions. The precise value is not critical so long as the resulting gray values are greater than zero and less than the maximum intensity. The histogram equalization step will adjust the resulting gray values across the dynamic range of the image giving a uniform result. In this work a value of three was empirically determined early on, and it has not needed to be changed for any environment or lighting conditions.

The hand image, even after feature enhancement with the normalized red channel, uses a narrow range of the intensity spectrum. The final step in preprocessing is to histogram equalize the pixels in the hand (ignoring the background). This spreads out the pixels across the available dynamic range, which further enhances the difference between light and dark regions (Figure 19d). It also compensates for images taken under somewhat

different light levels, since the brightest pixels in the hand will always be shifted to the greatest intensity values after equalization regardless of their level before it, and similarly for the darkest pixels.

Appearance versus model based recognition

A fundamental design decision was whether to build a 3D model of the hand during interpretation or to use appearance-based techniques. Building an accurate model of the human hand is a difficult problem. A hand has nearly 30 degrees of freedom, with mutual dependencies between many of them. Accurate estimates for each parameter can be difficult to extract even from ideal images because of problems like self occlusion, and in the real-world hand images are seldom ideal. The most common approach to reconstructing a hand model is to use iterative techniques with geometric constraints on hand shape as described in Chapter 2, which have little hope of running in real-time on current PC hardware.

Apart from the inherent speed and complexity issues of building a model, it is not clear that pose classification based on a hand model is superior to appearance based classification. While model-based techniques can potentially provide a good estimate of the shape of portions of the hand that are not visible, and so disambiguate poses that give the same appearance, intentional gesticulation is inherently a visual signal. People naturally use gestures that are easy to differentiate based only on appearance and display them in such a way that the important features are visible. This natural tendency is so strong that many times in the course of this work we observed users, without being prompted, redisplaying a misinterpreted pose to the system, but with the pose oriented more toward the camera and with the salient features exaggerated. This natural characteristic of people to display poses for easy interpretation makes an accurate estimate of the hidden parts of the hand less important.

Some hand models can actually make correct classification of visually distinct poses more difficult. A popular representation of hand shape in both mechanical and visual sensing work uses a list of the angles of the various joints of the hand. The problem is that some poses are very close to each other in joint-angle

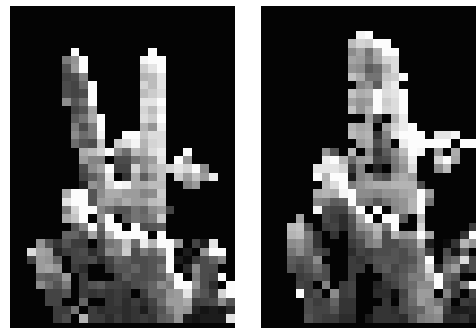


Figure 20: Two poses very similar in joint angle space, but easy to differentiate in image space.

space. For example, imagine two gestures with the middle and index fingers extended, the only difference that the two fingers are together or apart (Figure 20). With respect to joint angles these gestures only differ by a few degrees on one angle, the abduction between the two extended fingers. Since these poses are easy to differentiate, both visually by the receiver of the pose and tactilely by the person performing it, they naturally could have very different meanings. The dual to this example is a pair of poses that differ greatly in joint angle space but are difficult for people (even the person performing the pose) to tell apart. For example a grasp gesture could be either like holding a grapefruit or a peach. Between the two, almost every joint angle changes, some by half their range of motion, but the appearance is very similar, and the meaning could naturally be the same.

It is interesting to note the similarities between this argument and the work of Brooks in building reactive robots. In [Br91] Brooks argued very eloquently that the sequential model of intelligence — building an internal world model from sensor data, then examining the model to determine what actions to take — is flawed. His reactive robotic systems have shown interesting behavior with very limited internal models, using sensor input directly to guide action. There is evidence that the same is true in vision. Many successful vision systems have been built in industry for inspection, manipulation and process monitoring tasks [Kj96]. These systems generally use image features directly *without* building an explicit internal model. Such systems, however, are very specific to the task they are performing. The techniques they use do not generalize well to other problems. By training a network to respond as directly as possible to the sensor data without building an internal model, this work attempts to combine a model-less approach with some amount of flexibility.

Area versus outline based recognition

The vast majority of researchers use the outline of the hand for posture recognition (e.g., [Se92] [KH95] [Ma95] [NB95] [Hu95]). However in a realistic setting, especially when using local image cues for segmentation, the outline of the hand can be quite unreliable. Objects in the background are often segmented with the hand (Figure 21a). Portions of the hand can fail to be segmented for any number of reasons (Figure 21b).

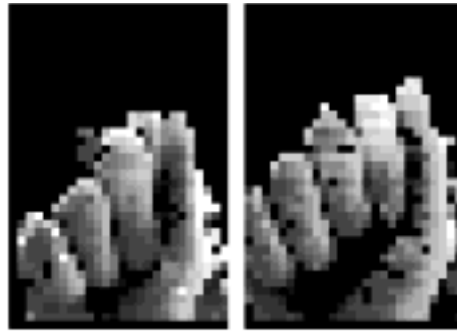


Figure 22: A pointing pose with the finger removed and a fist pose.

The entire hand region contains a great deal more redundant information about the pose than is found in just the outline. For example, one can count the folded fingers in front of the two poses shown in Figure 22 to determine that one must be a pointing pose with the extended finger missing, while the other is simply a fist. Clearly, if a pose recognition algorithm can make use of the information inside the area of the hand it will be more reliable than if relies on just the outline. By highlighting this internal data and providing it to the classification network, we have attempted to make use of this information. Tests described in Section 4.3.5 suggest the net has been able to make use of it for more reliable recognition. It should be pointed out that the cropped gray version of both images in Figure 21 and both images in Figure 22 were successfully classified by the networks used in this work.

Why a Neural Net?

The considerations described above limited our options for pose recognition. The argument for an appearance based approach, of course, eliminated any 3D modeling

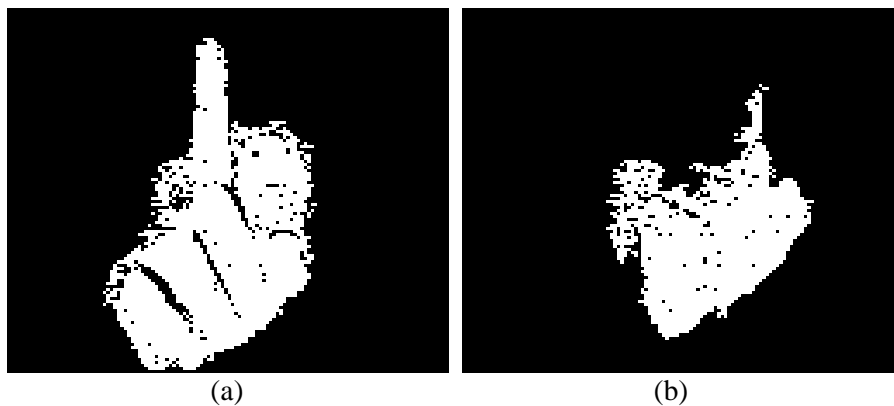


Figure 21: Two pointing poses with corrupted outlines.

scheme. The desire to use more than just the outline of the hand and real-time constraints argued against using a 2 dimensional model similar to those used in [HS95]. Special purpose hand coded feature extraction routines similar to those used by [Qu95] were considered, but another goal for the design was the ability of the pose recognition to be trainable to give the system flexibility. While this might have been possible using hand-coded feature extraction routines, it would have required designing a set of features which then could be combined in various ways to recognize different poses, a significant research topic in its own right.

This left the possibility of some type of matching scheme, where a pose image was classified by finding the most similar training example. The approaches described in Chapter 2, where an image is projected onto some basis to determine a vector in a feature-space, require quite a bit of run-time computational overhead that would have made real-time performance on current PC hardware unlikely.

A neural net has the capability to perform this type of match with very little run-time overhead. Very large training sets can be compiled down to the weights in the network, so that all the effort is expended during training. Networks are also very good at integrating multiple sources of errorful redundant data to reach a decision. This ability is potentially valuable when the segmentation is likely to be noisy, as it is here.

The approach used here was inspired by [Po92], where a preprocessed image of the road ahead was passed to a network to determine the steering direction for an autonomous vehicle. The network was trained by having it observe a human driving for some period of time. Recently, neural networks have also been used in a similar way for face detection [RBK96] and recognition [HB95].

Network Training

Training Set Variations After preprocessing, the pose classification scheme can assume the hand is centered in a known size image, that the background is primarily black, and that the variation in size within the image is small. It can also assume the range of rotations of the hand are small, due to the physical limitations of the user. However there is still some variation about all these dimensions, and with a finite training set, some views of the hand are bound to be underrepresented. Adding some variation along each of these dimensions to the images presented during training helps the network generalize to all possible views.

Two dimensional affine transformations applied to hand pose images approximate

some of the important variations found in the hand image. Translation approximates the effect of an improperly placed cropping window. Varying scale approximates the effect of users holding their hand closer or further from the screen. Rotating the image approximates rotation of the hand about an axis emanating from the camera (“yaw”). The other two rotation axes of rotation of the static hand pose roughly correspond to rotation of the hand about the axis formed by the forearm, referred to here as roll, and the rotation of the hand caused by straightening and bending the elbow or flopping the wrist forward and back, referred to here as pitch. The effects of these hand rotations will be discussed in the next paragraphs. The changes in appearance of the hand caused by rotation about the upper arm (i.e. swinging the arm to point to the left or right of the screen) can not be approximated by image warping, so this variation must be accounted for by variation in the training image set itself.

Empirically, rotation of the hand about the roll axis is minimal for most poses. While it is possible to rotate the forearm through nearly 90 degrees, when the hand is bent back, as it is in most poses, instead of being straight in line with the forearm, even small degrees of rotation require substantial muscle tension. As a result these poses are most comfortable within a very small range of rotations. The exception to this is with the pointing pose. There the hand is aligned with the forearm, and users tend to point with the hand rolled somewhere between the extremes of the palm (with its curled fingers) facing down and the hand rolled 90 degrees so that the palm faces to the left of the screen for right hand users (from the point of view of the user) (Figure 23). This roll changes the appearance of the pose in such a way that it can not be easily approximated by image transformations. To handle the range of roll the net must be trained with images that include a representative subset of that range.



Figure 23: Two extremes of roll in a pointing pose.

Pitch changes caused by elbow flexion do not naturally vary in this task, as the user sits at a relatively constant distance from the screen. This distance is determined by the need to type on the keyboard, typically placed directly in front of the screen, and it naturally determines the bend of the elbow. We have found in testing that when users are asked to perform a task without typing, they often sit much further back from the screen than they naturally would. This changes the pitch angle of their poses, and causes reduced accuracy during pose recognition.

More problematic are pitch changes caused by the wrist flopping forward or back. This causes foreshortening of the hand as the tips of extended fingers move toward or away from the screen. We have approximated this effect in the training data by having the trainer vary the angle of the hand and “point” the pose at different Y locations on the screen (even for non-pointing poses). It may be possible to approximate pitch by a perspective transform on the training data, but that possibility has not been explored.

Learning Protocols It can be hypothesized that the network would learn a better internal representation if first trained on noise-free data, then later introduced to the noisy output of the segmentation algorithm. This possibility was explored by adding a preliminary stage where the net was trained to correctly classify a set of 27 training images where background noise such as the face was manually removed, and segmentation errors such as missing fingers were restored. After learning to accurately classify this small set in about 100 epochs², the net was trained on the larger noisy training set described above.

This staged scheme did slightly reduce the overall learning time (in terms of total number of images presented to the network), but the final performance level on both the test sets and live data were not noticeably improved. The improvement in learning time was offset by the added complexity of this method, so it was abandoned.

The second step in the training algorithm used here, where the training includes images the network originally misclassified, effectively concentrates training on the areas that are most troublesome, namely those elements that are on the borderline of being classified as one pose or another, what Winston would refer to as the “near misses” [CF82]. This forces the network to more sharply define the boundary between classes, and is very effective at improving performance.

² An epoch is one complete cycle through the training set, each training image is modified once and presented to the network

3.6 Gesture Interpretation

At the top of the processing hierarchy lies a transition network (TN) which encodes the interaction language between the user and the system. As it parses the sequence of motions and poses performed by the user, the TN controls the rest of the system, determining when images are taken, what feature extraction operations are performed, and when actions are performed on the user's behalf. State transition diagrams similar to this have been used often to describe user interfaces. See [Ja83] for a discussion of the advantages of the technique, and pointers to work using similar representations. See Section 3.6.1 for a discussion of the merits of the representation for this domain. A slightly stylized version of the TN used in this work is shown in Figure 24. The complete TN is shown in text form in the Appendix.

Each path through the network that begins at the Start state and ends at either the Start

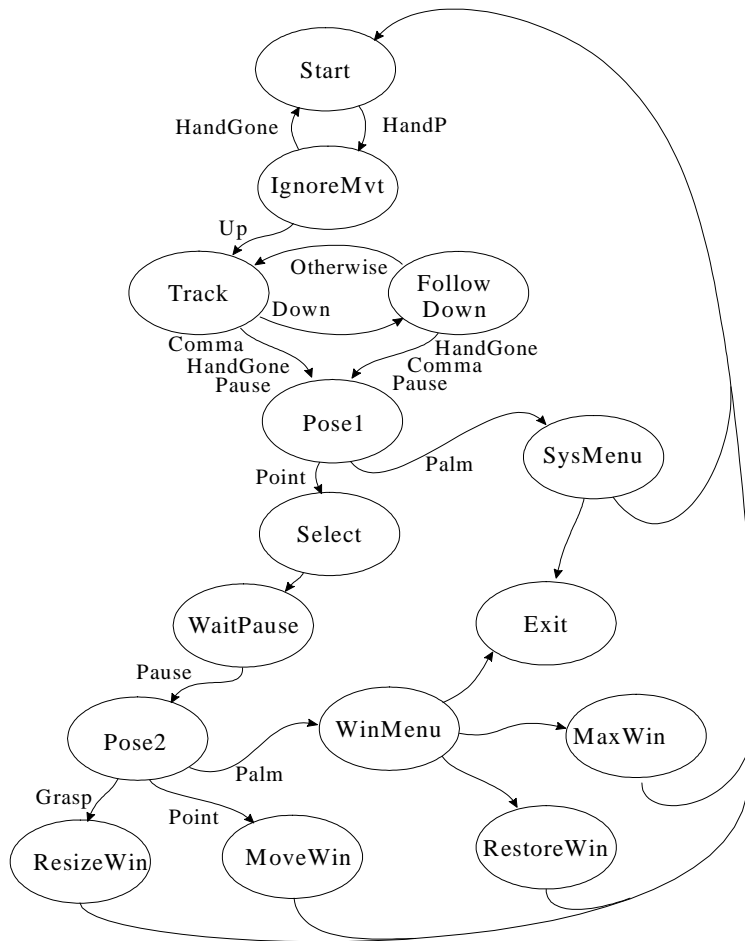


Figure 24: Transition network for window control task

or Exit states represents a different gestural sentence made up of a unique sequence of motions and poses. The system traverses the TN by extracting features from the image stream as described below. Every node can have actions associated with it, which are executed immediately on reaching that node. Actions are used to provide feedback to the user, perform some operation on their behalf, or record information for possible later use. The actions used in this work are shown in Table 2.

There are four types of transitions between states; *immediate* links, *menu* links, *pose* links and *motion feature* links. After a node is reached, and all its actions are executed, the system checks for an immediate link. If one exists, it is followed to the target node. If there is no immediate link, the system checks for any menu links (MENULINKs, in the terminology of the Appendix). If one or more exist, a menu is displayed with one item for each MENULINK. The user is allowed to make a selection as described below, and the system proceeds to the target node. If there are no MENULINKs, the system checks for any POSELINKs. If there are any, it segments out the hand from the image saved by the last RecordPose action, classifies it as described in Section 3.5, and proceeds to the target node.

Finally, if there are no POSELINKs, the system executes a tracking cycle. In other words, it records a new image of the user, undersamples and segments it as described in Section 3.2.3, determines the screen location as described in Section 3.3, smoothes the

Action	Description
RecordLocation	Saves the current location of the hand in register L
RecordPose	Saves the current image for later pose recognition
ShowCursor	Displays the cursor at the current hand location
SelectWindow	Activates the window closest to the location saved in L
MoveWindow	Moves the window closest to the location saved in L
ResizeWindow	Resizes the window closest to the location saved in L
MaximizeWindow	Maximizes the window closest to the location saved in L
RestoreWindow	Restores the window closest to the location saved in L to its former size (after a Maximize)
Exit	Stops the gesture interpretation process.

Table 2: The actions which can be performed at each node

path as described in Section 3.4 and then examines the path for the motion features specified by the LINKs from this node, as described in section 3.4.2. If none of the specified motion features are found, it follows any “Otherwise” LINK. If none exists, it continues to track the hand till it finds one of the specified motion features. It should be noted that if the system ever follows a link from a node to itself (e.g. the Otherwise LINK from the node Track1 in the Appendix), the actions for that node are executed anew. If there is no Otherwise LINK, and the system continues to track the hand for a number of cycles till it encounters one of the specified motion features, the actions are *not* executed each cycle.

Menus

When a node calls for a menu, a menu is displayed which has one item for each MENULINK from that node. The system then enters an interactive loop that allows the user to select a menu item by moving their hand up and down. During this time the characteristics of hand motion tracking are changed to make it easier to select individual items. Smoothing is increased by both increasing the mass of the cursor, and raising the knee on the sigmoid smoothing function, both of which help reduce noise in the cursor path at the cost of reduced responsiveness. Hysteresis is added between menu items so that the highlighted item “clicks in” and the system does not oscillate between two choices. When the desired item is highlighted, the user executes it by either dropping their hand to the keyboard in a continuous motion, by moving continuously back from the screen, or by executing a COMMA.

Currently menus are simple variations of the vertical menus found on most mouse-based systems. Items are displayed one above the other and vertical motion of the hand is used to select them. The items are drawn slightly larger than they usually are in a mouse-based system, 3/8" or 1cm tall. This arrangement has proven to be less than satisfactory, and alternatives are discussed in Section 5.2.2.

Language definition files

The transition network for an interaction language is saved in a text file like the one shown in the Appendix. When a language definition file is loaded, the system builds a C++ object for each node, link, menu, action and pose classification network (PCN). This process is relatively straight forward, except for the PCNs. A set of PCNs are built in advance, each one trained to differentiate some subset of hand poses. The topology and interconnection weights of each is saved in a file, along with descriptors for the poses

it has been trained to differentiate. When a grammar is being loaded, and the system encounters a node which branches based on hand pose, it searches for a PCN definition file that differentiates exactly those poses for which the node has POSELINKs. If a suitable PCN definition file is found, the network is created as defined in the file. Otherwise an error is flagged.

The interaction language

Two interaction languages were used during this work. The first used only the motion of the hand. The second used both motion and pose.

The motion-only language allows the user to select objects and/or bring up a menu to manipulate them. Screen objects can be selected by pointing at them, as described below for the full language. If the user selects an object, then retracts, say with a Comma, and points at it again, a menu of manipulation actions is presented. If the user selects the background a global menu is brought up. Figure 25 shows the transition network for this language. Since this was used only as an early test of the implementation, it will not be described in any more detail.

The complete interaction language uses both motion and pose (Figure 24 and the Appendix). It allows the user to select a screen object, then based on hand pose either

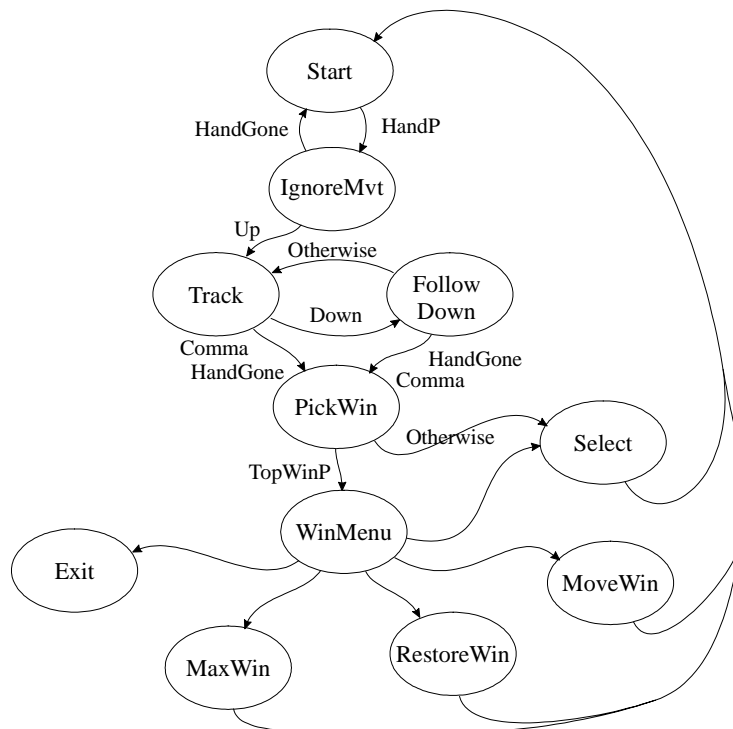


Figure 25: Interaction language using only motion features.

move it, resize it or bring up a menu of other operations. The user can also bring up a global menu.

The transition network (TN) leaves the start state when it identifies the hand in the image. The Track1 node tracks the hand, recording its location and drawing the cursor where the user is pointing. If the hand falls continuously down out of sight, the FollowDown node ensures that the position at the top of the trajectory is the one recorded. When the hand disappears, the screen object under the recorded position is brought to the front. If no object is under that location, the closest screen object within a small radius is selected. If nothing is within that radius, the motion is ignored.

If, instead of dropping their hand back to the keyboard, the user executes a Comma or pauses, the TN continues to Pose1 where the hand pose before the Comma (or during the pause) is classified as either a point or palm. A palm brings up a global system menu (SysMenu) which currently only gives the user the option to exit or continue. After the selection the system returns to the start state.

A pointing pose will select the object nearest where the user is pointing as described above, and the system waits for the Comma to end (WaitForPause) and then decodes the pose that occurs after it (Pose2). At this point the pose can be a pointing pose to move the object, a grasping pose to resize the object, or a palm to bring up an object-specific menu of operations (WinMenu).

Resizing and Moving a window, being interactive operations, have an inner loop that tracks the hand to determine the specifics of the operation. In the case of a move, the center of the window is placed where the user is pointing on the screen, and the hand is tracked this way until the user executes a Comma or drops their hand out of view. Just as with a menu, if the hand drops continuously down out of view the top of the trajectory is taken as the place to leave the window (This is not exactly true, for details see Section 3.7.6). Resizing is the same, except that the lower right hand corner of the selected window is tied to where the user is pointing. During both operations, the system increases cursor smoothing as it does for menus (see Section 3.7.4 for details).

At any time, the user can drop their hand to the keyboard and the system will return to the Start state, effectively resetting and waiting for new input. This way if the system misinterprets a user's gesture, they can immediately reset and resume.

3.6.1 Design Discussion

Language Representation

The transition networks used to represent the gesture interaction language are essentially augmented finite state machines. Specifically, registers have been added where information can be stored for later use, and actions can be performed when a node is reached. These modifications are similar to those used in Augmented Transition Networks (ATNs) [BF81].

It is possible to expand the transition network to be a true ATN by adding recursion as seen in Recursive Transition Networks, that is by allowing arcs to be labeled with names of sub-networks rather than just atomic “events”, and allowing these sub-routines to call themselves. The ability to use sub-network calls would allow some of the actions, like moving an object or bring up a menu, to be implemented in a more consistent manner. Currently they are self-contained interactive routines. It would be very natural to implement these as subnets of the transition network instead. As pointed out by Green in [Gr86], the addition of recursion would increase the descriptive power of the transition network — expanding the range of interaction styles it could support. These improvements have not been explored in this work, because the basic transition networks described here filled the needs of the interaction language. This is a good area for future work.

The transition networks have many similarities to the transition diagrams used in User Interface Management Systems (UIMSs) [Gr83][OI83]. UIMSs are systems that attempt to address the details of a user interface independently of the task which is to be performed. The goal is to have a single system that can be tuned to provide the interface for a variety of tasks. Some work in this area has used a transition network to represent the interaction with the user. Just as is done here, nodes represent the state of the interaction, arcs represent the events, such as mouse clicks, which can occur at each state, and actions can be associated with nodes or arcs. The focus in UIMSs is generally on issues such as how to best represent a large complex interaction in an application independent form and how to represent a large range of user activities. As such, the grammar is rich with structures such as sub-dialogs, semantic attributes and pervasive states. It is likely that a detailed study of UIMSs will be able to contribute to the design of a powerful transition network for hand gestures that can support a variety of tasks; however such issues are beyond the scope of this work.

In [Gr86], Green discusses the relative merits of using transition networks to represent

user interactions versus using grammars or event-driven processing. His conclusion is that the event model is the most descriptive model, and so may be best suited to form the basis of a user interface. It is interesting to note that an event model is not well suited to current vision-based gesture recognition systems. With these systems, the “events” that the user generates are features extracted from an image stream. It can not be ignored that many feature extraction operations are very expensive, such as the pose classification operation in this work. This is a significant difference from standard interface devices, where the detection of events (such as mouse clicks or keystrokes) is essentially free. At the least, this requires changes to the standard event model so that expensive events are only polled in particular semantic contexts.

It is also not clear when certain events occur in a gesture-based system. For example, as a user's hand changes shape from one pose to another, it could easily form a hand shape that would be considered a salient pose in some context. If a system were continually monitoring the hand shape for “pose-events”, it would report an erroneous event that may well confuse the system.

These examples point out that in some situations, standard HCI techniques must be modified to handle hand gesture interaction. The flexibility and continuous nature of gesture, and the expense involved in extracting gestural features, make hand gestures fundamentally different from most current interface devices, and interface programming practices will have to change to take that into account.

Hidden Markov Models

One obvious alternative to using a transition network to encode the interaction language is to train a Hidden Markov Model (HMM) to recognize it. HMMs have the ability to learn to recognize a pattern in a noisy token stream. One potential advantage of HMMs is that they can learn to recognize sequences in the presence of temporal variation. An HMM could be configured to use the motion and pose features extracted here as tokens.

In other work, Hidden Markov Models (HMMs) have been trained to recognize similar human actions. One of the earliest examples is described in [YOI92] where the Yamato et.al. train an HMM to recognize various tennis strokes (forehand, backhand, smash, etc.). The authors use simple image processing to create a sequence of binary images of a tennis player doing the stroke. Each image is turned into a feature vector and that vector is classified as one of several discrete exemplars. The sequence of exemplar vectors created during a stroke is used as the observed symbol sequence for the HMM. After

training several HMMs, one for each stroke, a stroke is classified by identifying the HMM most likely to have created its sequence of symbols. Recognition accuracy is very high under the constrained imaging conditions used. In another example, Starner and Pentland [SP95] use HMMs to recognize sign language. This work was described in Chapter 2.

HMMs were not pursued in this work for two reasons. The first problem is that this domain requires actions to be performed quite often during gesture recognition. These actions record data during the parse for use when responding to the user, they provide feedback to the user and they trigger expensive operations, such as pose recognition, which should not be executed when they are not needed. HMMs have no mechanism for this kind of intermediate feedback, the token sequence is classified only after it has completed. This would require either extending HMMs or using them to recognize the very short interactions that occur between feedback actions, which was deemed impractical.

A second problem with HMMs is that the interaction language is evolving rapidly. A transition network was chosen specifically because it is easy to understand and modify, so that various interaction techniques can be explored. An HMM would have required new training examples and retraining for even the smallest change. In this light, HMMs can be considered an optimization technique. Once the interaction language has stabilized, it is relatively easy to create a large set of training examples. The HMM would then learn to tolerate, even make use of the minor variations that might escape a human programmer building a transition network manually, making for more robust recognition.

Interface Design

Unlike a mouse-based interface, which has icons and menu headers on the screen, the gestural interaction language has no built-in cues to help the user remember how to perform an action. It is, therefore, important for the interaction language to be as easy to remember as possible. The gesture must have an easy to remember syntax and somehow “fit” the action to be performed.

To help achieve those goals, the interaction style used here was based on current theories of natural gesticulation [Mc92][Qu93]. One of their observations is that a gesture has three phases: preparation, stroke and retract. Preparation is the hand rising from its rest position toward where the meaningful portion of the gesture will begin, while simultaneously the hand begins to form the initial pose of the gesture. For the receiver of the gesture this essentially signals that a gesture is coming, and to attend to it.

The stroke is the portion of the gesture where the bulk of the meaning is conveyed. It is often preceded and followed by a pause where the pose is displayed to the audience. In most cases, the pose of the hand during the motion of the stroke is not meaningful. It may stay the same, or change to a pose that is again displayed at the end of the stroke. Finally a retraction occurs where the hand returns to its rest position, conveying to the receiver that the gesture has ended.

On the basis of this analysis, the interaction language is built around what we call a **core cycle**. In it, there is an initial motion where the hand assumes the position and pose of the gesture, followed by a pause to display the pose, a stroke to control the command and a retraction to end it. The core cycle is present twice in the longer path through the full motion and pose language. The first cycle selects the screen object or global menu on which to operate. The second cycle performs some operation on that object.

Generally the pose indicates which action the system is to perform, and the stroke controls that action. For some gestures, such as selecting a window, the stroke phase is vestigial and after posing the hand immediately retracts. For other gestures the hand is moved as a whole after the pose to convey additional information, e.g., to indicate the amount to shrink a window, before being retracted. For many gestures, the retraction is an important feature, indicating a salient event in the same way a button release does for a mouse. Thus it is important to find the location where the retraction began as accurately as possible. The way this is done will be discussed in detail in Section 3.7.6.

As the language evolved and incorporated modifications for usability and practicality it moved away from a pure Preparation/Stroke/Retract (PSR) cycle. A good example is the use of a Comma as a form of retraction, which does not really fit the definition but allows users to string together commands without returning to the keyboard between them. While the PSR cycle does help ensure a comfortable gestural rhythm, moving away from a strict adherence to it allows greater flexibility. Such a cycle is analogous to isolated word recognition in the field of speech understanding. It forces the user to separate each command manually by retracting their hand, which can lead to cumbersome interaction styles. It also limits the interaction language, eliminating many potentially promising gestures.

A second paradigm we use to help the user remember the interaction language is to give it a simple sentence-like structure. Each interaction can be considered a sentence with a subject, verb and possibly an adverb. First the user selects the subject of the sentence — the screen object to operate on. If the user returns to typing on the keyboard,

the implied verb is select. If the user instead Commas and strikes another pose, they are explicitly indicating the verb, or action they would like performed. Finally, the action may be modified by an adverb indicating where to move the window or how to resize it. Thus, typical sentences might be:

“That window, select”

“This window, move here”

“System Menu, that item, execute”

“That window, menu, this item, execute”

One version of the language allowed the user to string together action clauses by separating them with a Comma. Here the sentence could read:

“That window, select, resize like this, move here”

3.7 Implementation Details and Parameters

In any programming effort this size there are many parameters which control operation of the system, and many interesting but somewhat specific design decisions that must be made. This section documents the most interesting of these. Readers interested in the overall design and performance of the system may want to skip this section.

3.7.1 Hardware

The field of view of the camera limits the size of screen that the user can manipulate. A 3.5mm lens was the widest available for this work, giving a field of view of about 75 degrees vertically and 90 degrees horizontally. In order for this to provide good visibility of the hand no matter where the user is pointing, the screen used here is only 10.5 inches wide and 8 inches high.

The techniques used in this system will have no difficulty adapting to a lens with a wider field of view, even given the distortion it would introduce. The motion feature extraction operators look for qualitative aspects of the hand's path, and so should perform no worse in the presence of a gradual warping of that path that a wider angle lens would introduce. In addition, the polynomial screen mapping algorithm described in Section 3.3.1 would remove the majority of that distortion. The pose recognition routines currently reduce the resolution of the hand image before passing it to the PCN for

classification, so having the hand appear smaller in the original image is not a problem. The distortion of hand shape with a wider field of view should also not cause a problem, as the PCNs would be trained on images taken through the same lens.

The camera is angled up at about 40 degrees with the lens flush with the screen and about 2.5" below the bottom edge of the glass. This position images the primary work area very well, while hands on the keyboard are just out of view, which simplifies some aspects of recognition. Positioning the camera below the screen provides several other advantages in this domain, as were discussed in Section 3.1.1.

The tip of the user's finger leaves the top of the image when it gets closer than about 3" (horizontally) to the top of the screen. This means that if the user touches the top of the screen when pointing to it normally, most of the extended finger is cropped off. Ideally the camera should be tilted up slightly more and be closer to the bottom of the glass to provide a better view of the top of the screen. No important information would be lost at the bottom of the image. Unfortunately physical limitations of the setup did not allow that here.

The image processing hardware used was Data Translation's DT2871 Frame Grabber, which can operate at video rates, and DT2878 image processing board, containing an AT&T DSP32C DSP running at 40 MHz with 4 megabytes of memory. When new, this setup cost several thousand dollars. Equivalent functionality and power can now be obtained on a single board for several hundred dollars.

Images are transferred from the frame grabber to the DSP using a proprietary bus. While the theoretical transfer rate is higher, 15 frames/second was the best we were able to achieve. The transfer rate from DSP memory to host memory is considerably slower. This limitation had several effects on the design of the system that have been described elsewhere.

3.7.2 Segmentation

To calibrate the segmentation algorithms, three different templates are used corresponding to different hand shapes (Figure 26). These poses were selected as they image all surfaces of the hand expected during use at a range of orientations and lighting conditions. Each calibration pose contributes something different. The pointing pose emphasizes the back of the fingers when illuminated from the ceiling lights, and the extended finger often passes in front of ceiling lights, so the color predicate will be trained on any blooming that occurs. The fist pose captures the backs of fingers under a different orientation and the deep shadows formed by the fingers. The palm pose captures the color of the palm of the hand, which is often different from the color of the back.

The number of quantization steps used for each axis of the color predicate have a strong effect on segmentation performance. Coarser quantizations are faster to use, and generalize from the training data better. Finer quantizations are better able to differentiate subtle color variations, but require more training data and memory. To differentiate skin from the background with sufficient accuracy, hue and saturation need to be quantized relatively finely. Intensity, on the other hand, should be quantized more coarsely to improve generalization across different skin tones and lighting conditions. Empirically, 32 buckets for hue and saturation and three for intensity give good results in most conditions. In environments containing much color similar to skin-tone, it is helpful to increase the hue and saturation quantization to 64. Good results are generally obtained using a single bucket for intensity, but the system becomes sensitive to the dynamic range of the lighting. That is, images with strong light sources and dark shadows become difficult to segment accurately.

While training the color predicate, the Gaussian weighting function described in Section 3.2.2 is approximated by incrementing the count of the target cell by ten, cells in the 4-neighborhood by three and the remaining cells of the 8-neighborhood by one. For negative training examples the target cell is usually decremented by five and its 4



Figure 26: Three CP training templates.

neighbors by one, but this is manually increased to ten and three in difficult environments.

The morphological operations used during segmentation are designed to fill in and smooth the borders of regions by filling concavities and voids, and blunting sharp convexities, without changing the overall size of the regions. To achieve this effect, dilation operators use a threshold of 4. In other words, pixels must have at least 4 out of 8 immediate neighbors turned on, to be turned on itself. This fills in holes and acute concavities, but does not expand the overall size of the regions significantly. Erosion also uses a threshold of 4 (a pixel needs at least 4/8 “off” neighbors to be turned off). This shrinks sharp convexities, but has little effect on the overall object size.

3.7.3 Tracking

In order to get near-real-time performance on our hardware, we found it necessary to drastically reduce the resolution of the input image during tracking. Using a 64x60 image gave acceptable accuracy and tracking at rates up to 8 Hz. Higher resolutions help reduce the noise in the hand path, but significantly slow the tracking rate.

One problem during tracking is that if the cursor is placed at actual re-mapped location on the screen, it often appears behind the finger so the user has trouble seeing it. As a simple fix for this and to create the illusion that the fingertip is a laser pointer and the cursor appears where the beam would contact the screen, the cursor is drawn slightly above and to the left (for right handed users) of the actual computed location. The Y value of the displacement is scaled by the vertical location on the screen so that there is more offset at the top of the screen and less at the bottom.

3.7.4 Motion

The knee of the sigmoid force scaling function defaults to 150 (units are pixels). Increasing it much beyond 200 tends to make the cursor movement feel sluggish in that it does not keep up with moderately large movements well. Decreasing it below 100 starts to make the cursor twitchy on small movements, as small displacements elicit large forces on the cursor. The slope is typically set at 15. Increasing it beyond 25, making the curve flatter, again makes the cursor twitchy at small displacements. Decreasing it below 5, making the curve more like a step function, gives a noticeable step in the response of the cursor, so that it suddenly jumps after the hand when the knee is reached. Once set during system design, the knee and slope of the sigmoid have not needed to be modified.

The smoothing algorithm is designed so that the mass of the cursor is the primary tuning parameter. It is typically set at 6, and increased to 15 when a window is being moved. Increasing mass damps high frequency noise in the movement of the cursor, making it easier to precisely position the window. It also gives the user the illusion of weight, so it “feels” different when he is moving a window. The mass is also increased when the user is selecting an item from a menu, again to reduce jitter and help the user select the correct menu item. A lighter mass is desirable during general tracking as it increases sensitivity to small movements of the hand, so the cursor feels lighter and more responsive. During move and menu selection operations the knee of the sigmoid is also increased to an unrealistically large value so that the motion never kicks into “high gear”. This was found to be much more pleasing to use.

During menu selection the items are highlighted by reverse video as the user moves their hand up and down. We found it very helpful to flash the selected menu item several times after a user retracts and before it is executed so that the user received good feedback about which item was actually selected.

Certain motion feature extraction routines use two thresholds to approximate the speed of the motion. Movements below the lower threshold, currently 20 screen pixels/cycle, are classified as slow, between the two thresholds classified as medium and above the fast threshold, currently 100 pixels/second, considered fast. The system is relatively insensitive to the speed thresholds. Since originally being set, they have not been adjusted.

A separate threshold is used to determine when the hand is stopped. Given the constraints of this system, the best compromise in Pause detection was to look for the cursor moving less than 20 pixels in two consecutive cycles. This is adequate for a prototype, but faster tracking rates would allow the use of more reliable detection algorithms. This is discussed in more detail in Section 5.1.3.

Another threshold is the minimum size of the hand below which it is considered to have disappeared. Too low a threshold causes false negative retractions and too high a threshold causes false positive ones. Either case plays havoc with gesture interpretation. The system is moderately sensitive to the size threshold as under some lighting conditions the hand tends to segment larger than under others. A higher threshold also helps when there is a noisy background that might be confused for the hand.

Fortunately it is easy to adjust this threshold during system calibration. As the user is pointing at the screen to calibrate the image-to-screen warping function the smallest size

the hand blob reaches is recorded. At another time during calibration the size of the largest blob is recorded as the user is typing (so the hands are out of the image). A threshold is then chosen midway between these two. If these sizes overlap, an error is flagged. Generally values within 50 on either side of optimum produce good performance, giving a wide acceptable range, but occasionally the user must manually adjust the threshold for good performance.

Again, faster tracking would make it possible to reduce reliance on this threshold by using heuristics to check the path of the hand before it disappears. This is discussed in more detail in Section 5.1.3. Unfortunately, at the current tracking rate, the hand can disappear off the edge of the workspace from many places within it in one cycle, giving any heuristics very little information to work with. Other problems caused by a slow tracking rate will be discussed in more detail in Section 5.1.6.

3.7.5 Pose Recognition

The networks are standard feed-forward neural nets, trained using back propagation with momentum [Fu94]. During the initial debugging of the network code the learning rate was set to 0.1 for the entire network and the momentum term was set to 0.76. These values have not been adjusted since, leading to the assumption that network performance does not critically depend on their value, though no tests have been run to test this sensitivity.

During training each input image is modified in scale, rotation and translation. The random modifications form an even distribution within the following ranges: Scale plus or minus 20%; Rotation plus or minus 10 degrees; Translation plus or minus 5 pixels in X and Y independently (this equates to plus or minus 13% in Y and 19% in X).

3.7.6 Interaction Language Details

Since a very natural way to retract from a gesture is to simply drop the hand back to the keyboard, a problem arises in deciding what position was being indicated and what was the pose at the time. Ideally, the trajectory moves smoothly up to the intended target, pauses, then drops straight back down out of sight. In this case it is easy to determine the salient moment of the gesture and extract the location and pose information. Unfortunately this is not always the case. Often, the user quickly moves their hand up to the approximate location, then fine-tunes the position with several short movements before dropping their hand back to the keyboard. In this case, it is not possible to

determine the intended selection point till the hand drops down off the bottom of the image. This problem is addressed by adding the FollowDown node to the interaction language. The Track1 and FollowDown nodes form a loop such that as long as the hand is not moving downward, the Track1 node continuously records the position of the hand and saves an image from which to classify the pose. Once the hand starts to descend at medium speed the FollowDown node continues to track the hand without recording its location or pose until the downward motion slows or there is a non-downward motion. Then the net transitions back to the Track1 node and records the location and pose image again. This records the location and pose at the top of any trajectory that ends with the hand falling down off the screen for potential later use. This same algorithm is used during menu item selection. If the hand moves continuously downward and disappears, we execute the item at the beginning of the downward movement. As soon as any other movement occurs, we provisionally select a new item.

3.7.7 Window System Interface

Microsoft Windows™ 3.1 was used for several implementation related reasons. The device drivers for the frame grabber and DSP boards we had available were not available for competing systems such as OS/2 or Windows NT. The development environments available for Windows at the time were also much superior — an important consideration for a project of this size.

Unfortunately, the choice of Windows 3.1 had several ramifications. A major issue is that it is not possible to get access to the sub-windows of an application in order to manipulate them. To another program, all sub-windows of one application look the same, including windows that are no more than buttons or place holders and windows that are true sub-windows of the application. Neither can an application get access to icons on the desktop. Thus this work is limited to manipulating only the top level windows of an application, not sub-windows or icons. This, in turn, has limited the utility of the final system.

Another limitation is that Windows does not use pre-emptive multi-tasking in its scheduler. This required the code to explicitly give up control at key points in order to allow application programs to get access to the CPU. It also dictated a more complex system design — rather than having several communicating sub-tasks running in different threads, we had to use a single thread and explicitly switch between sub-tasks. All this complicated the implementation, and resulted in much more jerky switching between the

interface and application code than we would get under an operating system using a more sophisticated scheduler.