4  Performance Evaluation

The system has been used by the author during development and testing for hundreds of hours. It has also been used by about a dozen others for shorter amounts of time during demos and formal testing. Since it is as portable as any PC, it has been used in several different office and lab environments, without need for controlled lighting, prepared backgrounds or unusual care in its placement. It is not being used on a daily basis primarily because of its limited functionality. Some of the users have been given longer sessions with the system where different components were evaluated separately. The observations that follow are based on notes taken during these sessions.

4.1  Segmentation

Segmentation was evaluated qualitatively and by how well it supported the rest of the system. No quantitative metrics have been run. This section will describe our observations, discuss the algorithm's strengths and weaknesses, and in the process present enough examples of segmented images that the reader may get an idea of the quality of the segmentation.

4.1.1  Overall Performance

Quality of the segmentation varies depending on quality of calibration, environmental conditions, and the orientation/position of the hand within the scene. At its best, segmentation of the hand is near perfect in the sense that it identifies exactly those pixels in the hand against a black background. Often the segmentation is less than perfect, but accurate enough for smooth tracking and reliable pose recognition. Occasionally large
chunks of the hand are missing, it may break into many small pieces, or some other object than the hand is segmented instead (Figure 27). The percentages of these behaviors are roughly 15-20%, 80-85%, and <1% respectively.

The types of noise present in a typical segmentation come from several sources. Extended regions of the hand, including extended fingers, can be missing (Figure 28), especially when that region of the hand passes in front of a bright light source. The blooming or color shift in pixels near the edge of the hand affects segmentation in that region. The problem is most severe with extended fingers because of their thin cross-section. Additionally, holes can appear in the body of the hand due to deep shadows like those between the fingers in a fist (Figure 29). Since the camera in the system is facing back at the user, the face and bare arms can segment along with the hand (see Figure 30).

The current level of noise seems to cause only minor problems for the system as a whole. In spite of missing chunks, enough of the hand generally remains for reliable tracking. Since pose recognition uses the entire hand regions rather than relying on key features of its boundary, it does quite well in the face of missing or improperly added regions. A pointing pose without the extended finger has a somewhat greater chance of being classified as another pose, but it is still classified correctly the vast majority of the
time. This issue will be addressed again in the tracking and pose sections of this chapter (Sections 4.2 and 4.3).

Some of these problems can be corrected by the morphological operators. Properly tuned operators can split apart regions connected by long threads and reconnect regions that are nearly touching but are separated by a narrow canal, such that later processing steps, such as largest connected component analysis, will find a better segmentation. Even with optimally tuned parameters, however, some errors will always get through, so any modules using the segmentation must be able to deal with some amount of this noise. In particular, the boundary of the hand is often corrupted, which affects the choice of pose recognition algorithms that can be used. Various morphological steps were used in an earlier versions of the system but are not currently being used due to speed considerations. Their removal caused a barely noticeable drop in overall system performance.

4.1.2 Calibration

Good calibration is essential for good segmentation performance. Calibrating with one image often gives acceptable results but calibrating with 3–4 images covering the expected range of viewing conditions is needed for optimum performance. In difficult environments, for example if the background has much color similar to the user's skintone, it may be necessary to use a few more training images and to adjust the ratio of positive to negative update weights as described in Section 3.2.4 to get good performance. Only in the most difficult environments, such as where a skin colored wall lies directly behind the user or under some of the conditions described Section 4.1.3 will careful training and parameter adjustment not be able to produce a color predicate (CP) which performs well.

With novice users, calibration takes several minutes and can be quite frustrating because of the difficulty of placing the hand image within the training template outline. This problem was aggravated because the frame grabber board set did not allow us to show the user an inverted (mirror-like) image in real time, but only a direct camera view. Thus the user has to move their hand left for its image to move right. This inversion from the familiar mirror-like behavior made it much more difficult for novice users to align their hand with the template. We briefly experimented with aiming the calibration screen at 90 degrees to the user and using a mirror at 45 degrees to invert the image. This made it much easier for novices to calibrate the system, but its bulk forced us to abandon the
setup. With a little experience it becomes easy to calibrate using the camera view. An experienced user can train the CP in just a few seconds.

A color predicate can be saved to a file and reloaded. The typical pattern for the author was to load the same CP each day when beginning work with the system. The vast majority of time this would produce acceptable results, and only rarely was manual retraining needed. More than half the other users of the system were also able to get good performance from the system using the author's calibration file.

4.1.3 Performance in Different Environmental Conditions

Segmentation is very robust to environmental changes. Sufficient lighting is necessary, but nearly any well lit office qualifies. Most of the figures in this dissertation were taken with the screen in a corner of a typical office and ceiling lights providing illumination. From the point of view of the camera, the hands were lit only by indirect illumination from the walls and the faint light from the screen, as the lights were behind the user.

One potentially common situation that causes segmentation difficulty is when the screen is in a corner where the walls are very dark or shadowed by empty shelves and there are strong ceiling lights. In that case, very little light is reflected onto the hand from the walls, and the large intensity difference between the hand and the background causes the hand to appear very dark in the image. The low intensity makes the hue and saturation unreliable, in turn making segmentation unreliable. This has not been a very serious limitation in practice, as generally office walls are light colors.

When either the lighting is insufficient or the walls are very dark, simply adding a common unfocused desk lamp aiming down at the desk next to the monitor generally puts enough light on the screen-side of the hand to fix the problem.

Minor changes like a person standing nearby and shading a light, have little effect on segmentation quality. If the room lighting changes drastically, say turning on or off a ceiling light, retraining may be needed. Daily variation due to light coming in through a window can slowly degrade performance. Since this change is slow, it is possible to have several calibration files, trained at different times during the day, and switch between them when performance degrades. The system could likely be made to recognize when segmentation is degrading by monitoring metrics such as the number of segmented pixels scattered about the image (not a part of the large connected component) or watching for changes in the average size of the segmented region over time. It could then prompt the
user to recalibrate or try switching between different CPs.

The best performance to date has been achieved with the system in the center of a large, well-lit room, rather than being tucked into a dim corner as it usually is. The ceiling lights were hidden behind air diffusers and other clutter on the ceiling so that none were directly imaged by the camera. Being in the center of the room, the hand had good illumination on the camera side from lights in front of the user. Under these conditions, segmentation was nearly always excellent.

Most of the results presented in this thesis were obtained with the system in the corner of a normally lit office with light walls and ceiling lights that appeared in the image — good but less than ideal conditions.

4.1.4 Performance on Different Skin Tones

The system has been used with people having a wide range of skin-tones, including people of Asian, Indian, European and African descent, with equally good results. For any single skin-tone, a color predicate can be trained to provide good segmentation performance. Under good conditions it is also possible to train a CP using various skin-tones, so that it will work for a range of users. Training a CP to recognize a range of skin-tones does tend to increase background noise by a small amount. For optimum performance, each user should train the system before beginning work.

It is noteworthy how well the segmentation generalizes across different skin-tones. Because of the segmentation's emphasis of hue and saturation over intensity, a predicate trained on light skinned people often works well for dark complexioned blacks and all shades in between. The major exception to that is that CPs trained on Asians do not seem to generalize as well to others and vice versa.

4.1.5 Non-Hand Skin Regions

While skin regions of people in the background are generally identified by the CP, they have little effect on system performance due to their small size. Potentially more of a problem are the face and arms of the user.

Because of the wide angle lens used, the user's face appears significantly smaller than their hand. When both the face and hand are in the image, but the two are not in contact, the face is reliably ignored on the basis of size alone. When the hand is not in the image the face is reliably ignored by the minimum size threshold described in Section 3.7.4.
When the face is “connected” to the hand, it is segmented with it, producing a bulge on the border or filling in gaps between the fingers. Fortunately this has not proven to be a problem with either tracking or pose recognition. In the case of tracking, the relatively small area of the face does not significantly affect the centroid of the hand blob. Pose recognition is unaffected because it is based on gray level information inside the hand region rather than just the region border. While the face distorts the hand border, it does not affect details on the interior of the region (Figure 30a).

![Figure 30: Example of the face and arm extracted with the hand.](a) (b)

A user's bare arm is more of a problem because it is large, and almost always segmented with the hand (Figure 30b). This can cause problems tracking near the top of the screen, and makes the hand itself very small when the hand/arm is cropped for pose recognition. Because the shape of the arm and its location relative to the hand are very predictable, it should be possible to develop a fast algorithm to reliably eliminate it. However, for the purposes of this dissertation, users have been asked to wear long-sleeved shirts.

### 4.1.6 Other Issues Affecting Segmentation Quality

#### Screen Illumination

The glow from the screen has a different color signature than the ambient illumination, generally giving the hand a slightly blue tint when it is nearby. This can significantly affect segmentation results. A color predicate trained when the monitor is off will not perform as well when it is turned back on, and vice versa. The change in the color or quantity of screen illumination with different window configurations or background colors does not seem to be great enough to cause problems. A CP trained on one window configuration works just as well on nearly any other configuration. Since screen
illumination only affects objects within a few inches of the screen, beyond that ambient lighting overpowers it, the idea of using screen illumination to aid segmentation of nearby objects was briefly considered, but not explored further.

**Local Segmentation Irregularities**

Segmentation performance can vary quite a bit depending on where the hand is in the image. An example of this occurs when the user is pointing to the lower half of the upper left quadrant of the screen. While skin is generally Lambertian, under some conditions the specular component can still dominate the light received from a patch of skin. When the user is pointing to this region of the screen, the backs of their fingers specularly reflect a great deal of light from the ceiling. Since specular reflections generally retain the color of the incident light, and the color predicate has been trained to ignore the ceiling, portions of the backs of the fingers are lost.

Segmentation errors such as these are very repeatable, so they degrade system performance in predictable ways. With respect to tracking, as the hand moves through the region described above, the centroid deflects away from the backs of the fingers when they disappear, then returns as the hand continues on and they reappear. This is an example of the DC and Type 2 noise described in Section 3.4.3. The current perspective transform used for hand-to-screen mapping can not handle such local effects well, providing much of the motivation for switching to the improved scheme described in Section 3.3.1.

The effect of these local segmentation irregularities on pose recognition is mitigated by the fact that the pose classification networks are trained on real segmented data. Therefore, so long as the training data includes images that represent these irregularities, they have minimal effect. The exception to this occurs when the segmentation error makes one pose look very much like another, which can increase the probability that the two will be confused. This situation will be discussed in detail in Section 4.3.3.

**Examples**

Figure 31 shows some example pose images captured as part of the training set for the pose classification networks. These were taken with the user “aiming” the pose at different points on the screen in order to provide a range of appearances for each pose. You can see some of the problems which have been discussed here. The users face appears in several images, including palm-4, point-10, and grasp-14. Point-1 through point-9 were taken with the user pointing at the top of the screen. You can how the
extended finger was cropped by the top of the image. Palm-1 through palm-5 show the show the finger cropping and foreshortening that happens to the palm pose near the top of the screen. Point-5 and point-6 show the backs of the user's fingers being lost due to specular reflection from the ceiling lights. Several images, such as palm-3 and grasp-4, show a horizontal line across the top of the hand due to a code bug.
Figure 31: Example hand images from the PCN training set.
4.2 Hand Motion Tracking

In order to evaluate the ability of the system to track the user's hand, two objective tests were run. First the position of the cursor is compared to the raw hand location extracted from the image. This shows that the smoothing algorithm described in Section 3.4.1 eliminates the majority of the AC noise components, while faithfully reproducing the motion of the hand. Second, the ability of the user to select on-screen objects is evaluated by measuring the time it takes to place the cursor in an object on the screen. This shows that for objects larger than about an inch, selection time is comparable to that when using a mouse. Finally, the more qualitative aspects of tracking performance are evaluated by describing the users subjective impressions.

4.2.1 Smoothing Algorithm Performance

For good usability, it is essential that the cursor follow the hand accurately, especially during fast movements and changes of direction, but ignores the types noise described in Section 3.4.3. The key component in this ability is the smoothing algorithm described in Section 3.4.1. To observe the performance of the smoothing algorithm in controlled conditions, the location of the hand was plotted before and after smoothing. To observe the behavior under different dynamic conditions, the user pointed alternately at two objects, initially moving between them quickly, then moving more slowly each cycle.

Figure 32 shows a plot of the $x$ component of the position of a hand moving between targets on the right and left side of the screen. You can see that the smoothing algorithm

![Figure 32: Hand location before and after smoothing.](image-url)
effectively damps out the majority of the jitter that is present in the raw hand position data. This is most visible when the hand is still between about .5 and 2.5 seconds. It is also apparent that the cursor tracks fast movements very well. There is little or no delay when the hand begins to move suddenly, as you would see with a simple low-pass filter. This is seen most clearly on the sharp transitions around 3 and 4.5 seconds. When the hand stops moving abruptly, the cursor does not overshoot or oscillate around it (though the hand does occasionally overshoot the target). At around 9 seconds, the cursor does lag slightly behind the path of the hand. Here the initial movement was small enough to fall low on the knee of the sigmoid in the smoothing function, so the initial force on the cursor was small. The cursor caught up quickly, however, and then faithfully tracked the hand across the screen.

4.2.2 Object Selection Performance

Section 4.2.1 demonstrated only one aspect of the hand tracking performance, the ability of the system to faithfully follow the hand while ignoring the noise in the extracted position. From the point of view of a gestural interface, what is really important is the ability of the user to interact with objects on the screen. Doing this well depends not only on the ability of the system to track the hand accurately, but also on the ability of the user to identify screen locations using free-hand pointing.

This section evaluates object selection performance directly, measuring the length of time needed to select a target by pointing at it. To give this number context, the selection time is compared with that using a mouse. A mathematical model is developed to explain the results and used to predict the effects of improvements in tracking speed and accuracy.

The performance numbers presented in this section were obtained as follows. The user was seated in front of the system, close enough that pointing at any part of the screen was comfortable for them. The keyboard was placed directly below the monitor.

A trial consisted of the user pressing the spacebar key on the keyboard to trigger the drawing of target at a random location, then selecting that target as fast as possible. The target was initially a red circle 1 inch in diameter. The test was run in sets of 10 trials, with a short rest between each set. Having the user press a key to start the trial simulated the anticipated operational mode where object manipulations are interspersed with typing. To ensure that the user started from the keyboard at the beginning of each trial, they were asked to press the spacebar with their pointing hand.
Selection time was measured from the moment the key was pressed until the cursor had been inside the target continuously for 0.5 seconds. A time-in-target metric of success was used, rather than having the user select the object with a hand movement because the gestural selection mechanisms, Comma and Retraction are spatially noisy and would have made analysis of the results more difficult. This noise will be discussed in more detail in Section 4.4.2. Complete system performance, including the ability to select objects using gestural mechanisms will be evaluated in Section 4.4. Alternatives to this test, such as selecting the object using some non-gestural mechanism like a key press, were not explored. The 0.5 second time-in-target limit was imposed so that the user could not simply “fly” the cursor through the target, but had to keep the cursor inside it for a reasonable period.

The first test was performed on two users, one very experienced with the system (the author) and one experienced, but less so. Both were very familiar with mouse-based GUIs. Five sets of trials (50 selection operations in all) were run on each user. The mean selection time for free hand pointing was 1.91 seconds, with a standard deviation of .62.

The exact same test was run using the mouse to position the cursor, rather than free hand pointing. The mouse was a standard IBM PC mouse sitting on a pad wherever the user found comfortable to the side of the keyboard. The user was free to reposition the mouse to the center of the pad after each trial to avoid drifting off it. The mean selection time using the mouse was 1.57 seconds with a standard deviation of .17. These times include the .5 seconds within-target.

<table>
<thead>
<tr>
<th>Device</th>
<th>Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand</td>
<td>1.91</td>
<td>.62</td>
</tr>
<tr>
<td>Mouse</td>
<td>1.57</td>
<td>.17</td>
</tr>
</tbody>
</table>

Table 3: Selection times for 1 inch target with free-hand pointing.

The results of this first test indicate that a mouse has a distinct advantage over free-hand pointing for object selection, but it does not show the entire story. One attribute of free-hand pointing that quickly becomes obvious to a user is that as objects become smaller, they become significantly more difficult to select. While with most pointing devices selection difficulty depends slightly on target size [MK95], the effect is much more noticeable to the user with free hand pointing than it is with a mouse. To explore this effect objectively, the same test just described was run using targets of varying size. Five sets of ten trials were run at each target size. Here the only subject was the author.
The results of this test are shown in Figure 33 and Table 4. You can see that the time needed to select an object with free-hand pointing greatly increases with small objects, confirming the subjective observations. Selecting a 0.5 inch diameter circle takes about 3 seconds on average\(^3\). Selection time drops rapidly with increasing target size, leveling out at around 1.2 seconds when the circle reaches 2 inches in diameter. When the same test was performed using the mouse to position the cursor, selection time still varied inversely with the size of the target but the strength of the relationship was much less. Here the selection time dropped from 1.6 to 1.26 seconds.

<table>
<thead>
<tr>
<th>Target Size</th>
<th>Hand</th>
<th>Mouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>2.97</td>
<td>1.60</td>
</tr>
<tr>
<td>1</td>
<td>1.78</td>
<td>1.42</td>
</tr>
<tr>
<td>1.5</td>
<td>1.34</td>
<td>1.34</td>
</tr>
<tr>
<td>2</td>
<td>1.21</td>
<td>1.26</td>
</tr>
</tbody>
</table>

Table 4: Selection time in seconds versus target size in inches.

\(^3\) As with the previous tests, all times includes .5 seconds within-target time.
Analysis of selection time results

The greater standard deviation for free-hand selection, apparent in both sets of results, derives from an important characteristic of the current system. In some areas of the screen, selection is noticeably more difficult. Both components of noise, random jitter and offset, vary with screen location. The local variation in jitter has the effect that at two very localized screen locations, objects are difficult to select because of the random motion of the cursor. Because of the variation in offset, when a small object is in the upper right corner of the screen, it is nearly impossible to reach. Because some percentage of the targets are drawn in these areas during testing, a few trials have very long selection times (10 or more seconds) that drive up the standard deviation. Both the difficulty of selecting small objects and the increased standard deviation for them have ramifications for the design of a gesture system. This will be discussed further in Section 5.1.2 and 5.2.2.

Since the time to select larger objects with free-hand pointing and a mouse are about the same, what can explain the great increase in selection time for smaller objects? The increase is out of proportion with what would be expected based on research into free-hand selection in other domains [Fi54].

There are two important differences between free-hand pointing as implemented here, and most other pointing devices: the noise between the position the user thinks they are indicating on the screen, and the position where the cursor is actually drawn; and the delay (“lag”) between when the user moves their hand and when the system moves the cursor. The effect of lag on human-computer interactions has been studied recently in virtual reality applications [Br95][MW93], but the effect of random positional noise has apparently not been addressed.

The remainder of this section will model free-hand pointing, taking into account noise, lag and the inherent characteristics of human movement, and will demonstrate that these factors are sufficient to explain the results shown above. The model will then be used to suggest the limits of performance of free-hand pointing. We will begin by modeling the mouse results with standard techniques, then extend that model to explain the hand results.

Mouse Model

A great deal of HCI research has verified that Fitts' Law is a good predictor of the time needed for object selection for a wide range of pointing devices and selection tasks
(MacKenzie in [MW95], Zhui and Milgram in [ZM93], Card in [CEB87] and many others). Fitts’ Law states that the time to select an object of size \( W \) at distance \( D \) from the initial location of the cursor is given by

\[
T = a + b \log_2 (\frac{D}{W} + c)
\]

where \( a \), \( b \) and \( c \) are constants with values dependent on the details of the task and the pointing device.

Card uses this in what he calls a “Keystroke-Level Model” that attempts to model a range of different interactions by taking into account more details of the task, like the time for the hand to move to the pointing device (homing time) and time for different types of actions, such as keystrokes [CMN87]. Specifically, the model states that the time to execute an action \( (T_e) \) is:

\[
T_e = T_k + T_p + T_h + T_d + T_m + T_r
\]

where \( T_k \) is the time for all needed keystrokes, \( T_p \) is the time needed for pointing using a mouse or similar device, \( T_h \) is the “homing time” to get the user’s hand between devices, \( T_d \) is the time needed for drawing, \( T_m \) is the time needed for mentally preparing to act and \( T_r \) is the response time needed by the system.

A similar approach is used here to model the behavior of pointing with a mouse. In particular

\[
TM = T_h + T_p + T_t
\]

where \( T_h \) is the homing time (the time needed to get from the keyboard to the mouse), \( T_p \) is the pointing time (the time needed to move the mouse to the target), and \( T_t \) is the time-in-target. After [CMN87], \( T_h \) is taken to be a constant .4 seconds. \( T_t \) is a constant .5 seconds for this task. The pointing time is modeled by Fitts’ Law, where \( D \) is set to the expected value of the distance between the initial position of the mouse and the cursor, both random variables. \( D \) was approximated using half the width of the screen or 5.25 inches. The values of \( a \), \( b \) and \( c \) were first set to values found by others [CEB87], then manually adjusted within a reasonable range (as determined by Table 2 of [Ma95]) to fit the mouse data. This process resulted in

\[
T_p = .1 + .17(5.25/\mu + .5)
\]

Figure 34 shows \( TM = T_h + T_p + T_t \) plotted against the actual mouse data.
Hand Model

Following the same process with the hand data, it was not possible to come up with reasonable values of the constants to make the model explain the increase in selection time for small objects. When terms were added for lag and noise, however, a reasonable fit was possible.

As with most Fitts' law analyses, consider the problem in one dimension — that being the line connecting the hand and the target. This is clearly not completely accurate, as users were observed deviating from this direct route, and the noise occurs along two dimensions, but it will serve for an approximation. The user observes the position of the target and moves their hand (and so the cursor) toward it. Assume that at the end of some number of cycles the user is pointing exactly at the center of the target so that, in the absence of jitter, the cursor would also lie at its center. The total time to select the target now can be considered to have two components

\[ TH = T_F + T_n \]

where \( T_F \) is the time needed for the user to move their hand from its initial position to be pointing at the target, and \( T_n \) is the time needed for the cursor, in the presence of random noise, to actually land inside the target once the user is pointing at it. First consider \( T_F \).

\( T_F \) is approximated from Fitts' law by using the same values for \( a \), \( b \) and \( c \) as were
found modeling the mouse motion, with the exception that $b$ is modified to account for the lag in system response caused by slow tracking rate. Because a mouse and free-hand pointing have completely different physical characteristics, the actual values of the constants are likely to vary. For the purposes of this analysis, however, the mouse constants will serve as an approximation.

In [MW93], MacKenzie and Ware suggest that the effect of lag on selection performance can be modeled by adding its duration to $b$. In other words:

$$T = a + (b + LAG) \log_2 \left( \frac{\%}{\%} + c \right)$$

Following this model, and using the average cycle time for lag, gives

$$T_F = 1 + (0.17 + 0.14) \log_2 \left( \frac{\%}{\%} + 0.5 \right)$$

$D$ is set to the average distance the hand had to travel from the keyboard to when it was pointing at the target. After measuring the setup used in this experiment, $D$ was set to 7 inches.

Now consider $T_n$, the time for the cursor to actually hit the target once the user is pointing at it.

There are two main components to the noise in this domain, cursor jitter and absolute offset error. The jitter is a random variation in cursor position that occurs whether the hand is moving or not, and the offset is a relatively stable displacement between where the user thinks they are pointing and where the cursor actually appears (these errors were examined in detail in Section 3.4.3). As the target size decreases to approach the magnitude of these error components, the cursor is less likely to fall within it when the user first points at it, and less likely to stay within it for long enough to be counted a success.

It is possible to model the effect of jitter on selection time by estimating the probability that the cursor will land in a target in any one cycle given some amount of noise. It is much more difficult to model the effects of the offset error, as that causes the user to enter a feedback loop where they move their hand, wait to see the effect on cursor position, and move their hand again to correct for any offset. Rather than attempt to model this interaction, the model will only consider the random noise component.

Say the target has center $t$ and radius $r$. The cursor position is given by $c = t + \epsilon$, where $\epsilon$ is the jitter. Assume $\epsilon$ is random noise following a normal distribution with
mean \( t \) and standard deviation \( \sigma \). Then \( P_T \), the probability that the cursor will land within distance \( r \) of \( t \), is given by the area under the normal curve between \(-r\) and \( r \) [Fe71]. Table 5 shows some values of \( P_T \) for various values of \( \sigma \) and \( r \).

\( P_T \) is the probability of hitting the target in any one cycle. In the test described above, the cursor must stay within the target for .5 seconds, which is of course multiple cycles. During the course of this experiment the tracking rate averaged about 7.1 Hz or .14 mS/cycle, but the system does not run on a fixed clock, and individual cycles have been seen to vary from .1 to .2 mS. To achieve .5 seconds in the target, then the cursor would have to be inside it for from 3 to 5 cycles, averaging about 3.5 cycles. What must be determined is how long the system will have to wait for the cursor to land inside the target for that number of consecutive cycles.

Say the cursor must land in the target for \( n \) cycles. It is possible to determine the probability that cycle \( i \) will be the first time that has occurred \( (P_i(i)) \). Clearly, for the first \( n-1 \) cycles \( P_i(i) = 0 \). \( P_i(n) \) is simply the probability that the cursor was within the target for the first \( n \) consecutive cycles, or \( P_T^n \). The probability that the \((n+1)\)-th cycle will bring success is the probability that the cursor has been within the target \( n \) consecutive cycles, times the probability that the \( n \)th cycle did not bring success, or

\[
P_i(n+1) = P_T^n (1 - P_T^n)
\]

In general, the probability of success in any cycle is given by

\[
P_i(i) = P_T^n (1 - P_T^n)^{i-n}.
\]

The expected value of this probability distribution will give the mean number of cycles to successfully “select” the target when the user is pointing at its center, at a given \( P_T \).
<table>
<thead>
<tr>
<th>Std Dev $\varepsilon$</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>30.6</td>
<td>6.4</td>
<td>4.2</td>
<td>3.7</td>
</tr>
<tr>
<td>0.45</td>
<td>24.1</td>
<td>5.5</td>
<td>4</td>
<td>3.6</td>
</tr>
<tr>
<td>0.4</td>
<td>15.8</td>
<td>4.8</td>
<td>3.8</td>
<td>3.5</td>
</tr>
<tr>
<td>0.35</td>
<td>12.5</td>
<td>4.3</td>
<td>3.6</td>
<td>3.5</td>
</tr>
<tr>
<td>0.3</td>
<td>8.8</td>
<td>4.0</td>
<td>3.5</td>
<td>3.5</td>
</tr>
<tr>
<td>0.25</td>
<td>6.2</td>
<td>3.7</td>
<td>3.5</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Table 6: Expected number of cycles it will take for the cursor to land inside the target for 3.5 consecutive cycles at various levels of noise.

Because of problems dealing with the first $n$ cycles when $n$ is not an integer, we approximated the expected value by averaging the expected values at $n=3$ and $n=4$. Table 6 shows the results, in other words the expected number of cycles it will take for a noisy cursor to land in a target for 3.5 consecutive cycles. From this, it is straightforward to compute $T_n$ for any target size and expected amplitude of noise by multiplying by the average cycle time.

$TH = T_r + T_n$ can now be used to estimate the time to select a target on the screen using free-hand pointing. Setting $\sigma$ (the standard deviation of $\varepsilon$) to be .35, and plotting
the estimated values against the actual hand selection data gives Figure 35. The fit is not perfect, so there may be other factors at work, but the model seems to be a good first-order approximation of the observations.

This shows that the random noise and lag in the cursor position can have the effect, seen in the experimental data, of greatly increased selection time for small objects. As the radius of the target shrinks below the standard deviation of the amplitude of jitter, the selection time begins to increase rapidly. For objects larger than about two standard deviations, selection time using free-hand pointing using this prototype system is comparable to that of a mouse.

**Predicting the Effect of Performance Improvements**

Given a model that seems to explain the data, it can be used to answer some questions about the system. In order to improve performance on small objects, clearly one effective approach is to reduce the amplitude of the noise in the cursor position. Intuition says that another approach is to increase the tracking rate. The remainder of this section will predict the effect of these changes using the model derived above.

First, to explore the effect of increasing tracking rate, the rate used to estimate lag and $T_n$ was doubled. This had the unexpected result of actually increasing the predicted selection time for small objects, while the selection time for larger objects improves slightly (Figure 36).

![Figure 36: Predicted selection time from simply increasing tracking rate.](image-url)
On closer examination, it turns out that the increased selection time is an artifact of the test being modeled, where the cursor must be within the target for a fixed time. By doubling the clock rate, the number of consecutive cycles the cursor must remain in the target doubles. At a fixed level of noise, the probability of this happening drops faster than is compensated for by the increased clock rate. It is not clear how this will generalize to performance on other tests, but it does suggest that simply increasing tracking rate is not a magic bullet for better performance.

To explore the effect of reducing the noise in the cursor location, the selection time performance was predicted using the current system speed (7 Hz) but with the noise amplitude ($\sigma$) reduced from .35 to .25 and with no noise ($T_n=.5$). The result is shown in Figure 37.

The model predicts the selection time for a .5 inch diameter object will be .7 seconds faster by reducing the noise this modest amount (reducing $\sigma$ from .35 to .25). It also shows that even with no positioning noise, selection time for large objects will not improve at the current clock speed.

Figure 37: Predicted free-hand selection time with a reduced level of random noise ($\sigma=.25$) and with no noise ($T_n=T_t$).
In a system like this, operating near its limits of performance, it is not realistic to improve both speed and noise. Here, noise can be reduced by increasing the resolution of the tracking window — giving a larger hand blob, in turn giving a statistically more stable estimate of the center of the hand — as well as by using more expensive algorithms during segmentation and path smoothing. These will slow the tracking rate, however. To explore the trade-off between tracking rate and noise, performance was plotted at two levels of decreased $\sigma$, but with corresponding decreases in tracking speed. Figure 38 shows plots for tracking at 6 Hz with $\sigma=0.25$, and at 4 Hz with $\sigma=0.15$. These numbers were chosen arbitrarily, and do not represent estimates of the actual relationship between speed and noise.

From the plot in Figure 38 you can see that there is indeed a trade-off. At 6 Hz, selection time for small objects is significantly better, but for larger objects, shows a slight increase. At 4 Hz the selection time for large objects has again increased, and the increase in selection time caused by the slow tracking rate has actually offset the effect of decreased noise on small objects. For any practical system, this trade-off must be optimized for the intended tasks.

<table>
<thead>
<tr>
<th>Target Size (inches)</th>
<th>Selection Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.51</td>
</tr>
<tr>
<td>0.5</td>
<td>0.52</td>
</tr>
<tr>
<td>1</td>
<td>0.52</td>
</tr>
<tr>
<td>1.5</td>
<td>0.53</td>
</tr>
<tr>
<td>2</td>
<td>0.54</td>
</tr>
<tr>
<td>2.5</td>
<td>0.55</td>
</tr>
<tr>
<td>3</td>
<td>0.56</td>
</tr>
<tr>
<td>3.5</td>
<td>0.57</td>
</tr>
</tbody>
</table>

6 Hz, Low Noise
4 Hz, Lower Noise
Hand
Mouse

Figure 38: Selection time performance for realistic targets of tracking rate and noise.
Finally, to get an idea of the best possible performance of free-hand pointing, the model was set for no noise and fast tracking. The result, shown in Figure 39, predicts that with a realistically fast tracking rate, selection time using free-hand pointing can be expected to be approximately the same as for a mouse, slightly better for large objects, and slightly worse for small ones. Under ideal conditions (i.e. no tracking lag at all), gesture has the potential to be significantly faster than using a mouse for objects of all sizes.

This result should not be surprising when the task is examined closely. Mousing requires the hand to travel to the mouse (homing time), then move the mouse a small distance to position the cursor. Free-hand selection requires moving the hand about the same distance as it must travel when homing for the mouse, but the object to be selected is generally smaller than is the mouse. Therefore, you would expect free-hand object selection to take, at best, slightly longer than the mouse homing time, but less than homing time, plus mouse object selection. This is exactly what the model predicts. Removing the .5 seconds time-in-target, it predicts selecting a 2 inch object will take about .5 seconds, versus mouse homing time of .4 seconds, and total mouse selection time of .76 seconds.

This section has described the performance of the system with respect to free-hand object selection, and presented a model that explains the experimental data. The model
shows that random jitter in the cursor position and the lag caused by the slow tracking rate are sufficient to cause the long selection times for small objects. Predictions based on the model suggest that selection times can be reduced significantly on small objects by reducing the amplitude of the jitter, while improvements in tracking rate will have less of an impact on performance. Finally, both the model and an analysis of the task suggest that under ideal conditions, object selection with free-hand pointing has the potential to be faster than with a mouse.

4.2.3 Subjective Evaluation of Tracking Performance

Tracking accuracy and object selection times are objective measures, but they do not convey a complete picture of how well the user can interact with screen objects. This section presents some subjective commentary on tracking performance from the point of view of a user.

The cursor motion feels natural, and gives a good illusion of following where your finger is aiming. Some users initially have trouble, because they tend to point by aiming their fingertip, while leaving the hand in one place. Once they realize the problem, they generally have no trouble placing the cursor where they want it.

At some screen locations the cursor wanders noticeably with respect to the fingertip when the hand is still. The amplitude and frequency of this motion are such that it is not terribly distracting. Small displacements are followed immediately, though slowly, making it easy to position the cursor precisely with small hand movements. On larger movements an attentive user can sometimes notice a brief lag, after which the cursor tracks the hand very well. Generally this lag is not noticeable, because on fast movements the user is looking toward the target of the movement and the cursor seems to get there as soon as the hand does.

Tracking performance is noticeably worse near the top corners of the screen, primarily due to DC noise which can make it difficult to reach a location. Depending on environmental conditions, isolated screen locations can have trouble with Type 3 noise, appearing as 1-2 inch jumps 2-3 times per second. This is rare enough not to be an issue in a prototype, but it is an issue that must be addressed for any real application.

Selecting large objects by pointing at them is easy and very natural. This generally applies to windows, except when they are almost completely covered. Tasks like selecting two windows alternately to type into them are much more comfortable using gesture than using the mouse. Selection of icon-sized objects is noticeably more difficult,
while menus and toolbar items are small enough to be a challenge. Accurately identifying character-sized objects is impractical.

When the mass of the cursor is increased, as when moving a window, the user definitely gets the feeling that the object is “heavier” as it lags noticeably behind the hand and is quite stable when the hand is not moving. This feels quite appropriate for operations like moving a window. It is a bit cumbersome for selecting menu items, however.

For a time a touch screen was setup in the lab that was connected as a direct mouse replacement in MS Windows 3.1, and was used by the author for various tasks. It is interesting to compare subjective impressions of gesture versus the touch screen. While the touch screen in general feels quite precise, it is still difficult to select small objects. Here the cause is that the system is using a select-on-first-touch strategy, so user can not fine tune the cursor position before the object is selected. It was also annoying that as the size of the object approaches the size of the fingertip, the finger hides the object, making it harder to hit. As a result, just as with gesture, selecting small objects becomes difficult. Sears and Shneiderman address some of these issues in [SS91].

4.3 Pose Recognition

The neural networks described in Section 3.5 can be trained to classify segmented hand pose images correctly near 95% of the time. This section will discuss classification performance in detail and examine how it is achieved. The section begins by discussing the issues involved in evaluating network performance. It then focuses on what is needed to train the networks for best performance, and what factors can affect the performance achieved. The internal weights of a trained network are examined in an attempt to determine what features the nets use for their classification. Finally some variations are examined including changes to the number of hidden units in the network, having the net classify binary rather than gray hand images and using Boolean rather than multi-pose classification networks.

4.3.1 Evaluating Network Performance

The primary interaction language used two pose recognition networks. The first only needed to differentiate between a raised palm and a pointing hand. The second network classified three poses, a palm, a point and a grabbing pose, as if the user were holding a
large invisible softball (Figure 40). The experiments described here refer to the 3-pose network.

![Figure 40: Examples of the three pose classes differentiated by one of the PCNs.](image)

Each time a network is trained, even if the same set of training images is used, its performance will vary slightly. This is a result of several random variables used during training. These are used to determine the initial weights of the net, the order in which the training images are presented, and the amount by which an image is modified each time it is presented. Many of the results presented here are given in terms of the range of values observed over several trials.

Performance of the pose recognition networks has been tested in three ways: using a static set of hand images, testing interactively with the user, and evaluation in the context of the complete system.

A static set of test images is the primary measure of performance. It is difficult to get truly representative data in a static image set. One problem is that a user often tends to form his hand differently when forming a pose in isolation than he does while dynamically gesticulating. This is caused both by what would be called “co-articulation effects” in the speech community, and also by the user concentrating on forming the pose correctly rather than concentrating on the task at hand. A second problem is caused by the large number of possible views of the hand caused by the combinatorics of viewpoint, minor variations in hand shape, local variations in segmentation quality, lighting and numerous other factors.

To attempt to obtain a realistic amount of variation in the training set, pose images were captured in context as much as possible. For example the trainer might point to a window, pull their hand back then come forward again forming the grasping pose as if to resize it, then pause and strike the space bar with their other hand to capture the image of the grasping pose. In addition, the trainer attempted to form poses with some amount of
variation in shape each time, and poses were "aimed" at the points of a grid as described in Section 3.5.

In spite of our attempts to get a representative set of images, there always seem to be images obtained while the system is in operation that look significantly different from any in the image sets. This was addressed during training by adding incorrectly classified images to the training set as described Section 3.5. During testing, however, the limitations of a static test set can show overly optimistic performance numbers. Therefore, the nets were also tested interactively by sitting a user in front of the system and having them form poses for the network to classify. Because of its very nature, numbers obtained from interactive testing are imprecise. Indeed, the user can affect the results by intentionally presenting the system poses in ways that make them either easy or difficult to classify. The results of interactive testing that are presented here represent typical numbers observed over several testing sessions, and with the user (the author) attempting to be fair and realistic in how the poses were presented, but the reader should note the obvious opportunity for unconscious bias.

### 4.3.2 Network Training

Our experience matches that of others working with neural nets in that good network performance depends on having a large and representative training set. Initial experiments with some 30 base images (recall that the base images are modified in translation, rotation and scale during training) resulted in only about 80% accuracy when tested on live data. Increasing the size and representativeness of the training set was the largest single factor in improving classification performance. The results described here are based primarily on two disjoint sets of 120 images (40 of each of three poses). Typically one image set was used for training a network, the other image set was used for testing it. Figure 31 shows some examples from one of the sets. Adding the second stage of training, which includes the images misclassified after the first stage, improves performance still further.

Currently the system is trained on the initial training set for about 300 epochs, or 36,000 image presentations, during the first stage of training. Figure 41 shows how performance progresses during this stage in a typical training run (the graph has been smoothed very slightly to make it more readable). Notice that performance on the test set lags somewhat behind that of the training set, but catches up after 200 or so cycles.

The network resulting from this first step correctly classifies the test set 90% - 93% of
the time, with the errors spread roughly evenly across the various poses. When tested interactively, however, this single stage training only produces between 80% and 85% accuracy with the majority of the errors occurring on the highly variable pointing pose. For the second phase of training 30 or so misclassified images are collected during interactive testing. Since these are not distributed evenly across the poses, they are padded with correctly classified images so that there are an equal number of new images of each pose. These new images are added to the original training set and the network is again trained from scratch for 300 epochs. The resulting net has about the same range of performance on the test set, but performance during interactive testing is significantly improved, often topping 95%.

4.3.3 Sources of Error

Location Dependencies

Misclassifications often seem to be clustered in particular areas of the workspace. This has the annoying effect that a pose will be regularly misclassified during some operations, and almost never during others, making particular actions annoyingly difficult for a user to perform. Careful training can help reduce the severity of the problem. Taking ample training images from the difficult areas during the second phase of training helps the net learn to deal with some local irregularities. This helps a great deal with errors where the hand is segmented largely intact, but there are spurious or missing regions. The networks can take advantage of the redundancy in the gray images to produce a correct result the majority of the time.

Figure 41: Total classification performance versus training cycle for the training and test sets.
Some regions of the screen, however, have multiple problems that reduce the amount of information present in the image sufficiently to make accurate results very difficult. For example, when the user is pointing to the top of the screen, the extended finger is often lost off the top of the image. The remainder of the hand is distorted by foreshortening and the hand is often surrounded by ceiling lights, so that large pieces are missing and the gray scale detail is washed out. The result can look much more like a palm pose than a point. In these cases, no amount of training will allow the network to perform well, and the only alternative is to change the imaging conditions to improve segmentation and image quality.

Currently, the most troublesome regions are the extreme top and left edges of the screen. For this prototype system, the field of view of the camera was limited by the widest angle lens that was easily available. For a production system, a custom wide angle lens could be manufactured. The amount the camera is tilted toward the top of the screen was also limited by the design of the screens that were available for this work. This resulted in an image with more area on the bottom than was needed, and less on top. In a production system, the camera could be better integrated into the monitor to provide a better view of the top of the screen. Fortunately, most of the activity with the system takes place near the center of the screen, so errors near the very top and left edge do not affect usability significantly.

**User Dependencies**

Because the pose recognition networks (PCNs) take a fair amount of effort to train, an important metric is how well the PCNs are able to classify the poses of one user after they have been trained by another. There is a surprising range of ways one can hold their hand while pointing or when told to “grab a softball”. Indeed, a user often shapes their hand differently in one circumstance than another (say being asked to point at the screen for training, then pointing naturally while selecting a window). This naturally works against good cross user performance. Our experiments indicate that most users form their poses differently enough to drop pose recognition performance to around 80%, in some cases performance drops significantly more. The user could be trained to form their poses to give good performance, but that training is laborious and against the spirit of an adaptive system.

Ideally, we would want a PCN to be able to recognize all variations of hand poses equally well. To this end, we have attempted to train a network using training examples from multiple users. Initial results suggest that these networks are much better at reliably
recognizing poses from them all. This suggests the obvious — that a system intended to be used by a range of people should be trained on images from as many people as possible. Practical considerations limited us from doing that in any realistic way.

Environmental Dependencies

Variations in lighting can affect pose recognition performance. For example, side lighting will put a different pattern of light and dark regions across the interior of the hand image than lighting from above. Different lighting conditions also affect the pattern of segmentation errors. This was most evident when moving the system from one location to another. If the lighting conditions were similar, pose recognition was unaffected. If it was significantly different, say one office had a large window and the other did not, performance would drop by some 10%. This suggests training data for the PCNs should come from a representative range of lighting conditions. Some of this may be able to be simulated in the training data as Pomerleau did with passing cars [Po92]. No attempt has been made to do this here.

4.3.4 Network Weight Analysis

Recall the network architecture described in Section 3.5, where each pixel of the input image is passed to an input unit, the input units are completely connected to each of 20 hidden units, and each hidden unit is completely connected to a set of output units, one for each possible pose classification. This architecture can be considered to use the hidden units to find features in the input image that the output units can then use to differentiate between the poses. The features that a hidden unit extracts from the image are encoded in the weights between the input units and each input unit.

Figure 42 shows those weights in such a way as to try to make the features visible. These results are from a net with 20 hidden units trained to recognize three poses. The top four lines of images show the weights between the input image and each hidden unit. Bright pixels are those in the input image that excite the hidden unit while dark pixels suppress it. The bottom lines show the weights between the hidden units and the 3 output units. Again light shades represent strong excitation and dark represents strong inhibition. Here, the blocks to the left represent the connection weights from hidden unit 1, 2, etc. and those on the right from hidden unit 19 and 20.

A close examination of the weight patterns, as well as observations of the behavior of the hidden units while several dozen different pose images were presented to the net,
Some of the patterns identified by the hidden units seem to correspond well to features humans would recognize in an image. Notice the finger-like projection in units 5, 8 and 10. Testing revealed that these do indeed respond to pointing images with extended fingers in those positions. Many of the features are sharp and distinct, but not easily identifiable, such as the patterns in units 7, 15 or 19. Some hidden units appear that they should respond to individual poses, e.g. unit 9 seems to have a faint pattern of a complete palm pose, fingers and all — however it responds most strongly to a spread pose, and very little to a palm.

Some units seem to correspond to salient features, but really do not. Units 2 and 18 appear to be looking for a fist of fingers in a particular orientation, very similar in shape to (but horizontally mirrored from) how they appear during some pointing poses (Figure 43a). Interestingly, there are no images in the training set where the fingers lie in the position and orientation apparently seen in those nodes. Also, notice that both units
Figure 43: Example images for network weights discussion.  a) an image similar to what we would expect hidden units 2 or 18 to respond to, b) an image that unit 2 actually does respond to, c) the type of image unit 6 responds strongly to, d) the type of image that excites units 9 and 11.

excite a palm interpretation and are neutral about a pointing pose.  When various images were applied to the net, unit 2 responded to very few images, almost always producing a 0 output.  Occasionally, however, it would respond very strongly (output .98), while no other hidden unit responded at all (output 0).  The images it responded to were always similar to Figure 43b, a palm pose where the hand is near the top of the image so there is strong foreshortening and the tops of the fingers have been cropped by the top of the original image.  When unit 2 does respond in this way, it reliably produces a palm interpretation.

Some mysterious patterns appear between hidden unit features.  Notice that units 17-20 seem to be the same patterns as units 1-4, except that they are shifted down and to the right.  These may be looking for the same poses which are translated by some amount within the image.  The cause of features such as the faint bar across the top of hidden unit 17 is unknown, but they may be due to systematic patterns produced during modification of the training images.

Finally, it is noteworthy that some hidden units seem to do most of the work for a particular interpretation.  For example, unit 6 is very important for recognizing palm poses.  It responds very strongly to typical palm images (such as Figure 43c), while most other hidden units respond very weakly.  Units 9 and 11 seem to most of the work of recognizing spread (or grasping) poses (Figure 43d).
4.3.5 Variations

Binary hand images

An assumption made during the design of the pose classification algorithms was that the networks would be able to better classify poses if they had access to the full gray level image rather than just to the outline of the hand. To determine if the networks are making use of the information contained in the gray levels, a network was trained as described above but the pixels of the training and testing images were thresholded at a low level to produce images where every pixel in the hand was white and every pixel outside it was black. Figure 44 shows the learning curve for one such network. Notice that the X axis has a different scale than the earlier plots, corresponding to longer training.
The results show that the networks clearly make use of the gray level data to improve performance. With binary data the nets took much longer to learn the task and they never learned it to the level of proficiency achieved by the nets using gray level data, with performance leveling out in the mid-80% range rather than the mid-90% range on the training set. These “binary” nets also do not seem to be able to generalize as well as nets using gray scale data. The performance on the test set only reaches around 70%. This is also seen in interactive testing, as performance of the binary nets runs 60-70%, significantly worse than with the training/test sets, while the “gray” nets perform at a level comparable to the static training/test sets. Finally, the learning curve for binary data is much more erratic. Figure 45 shows the unsmoothed performance on the palm gestures in the test set. The highly non-monotonic curve makes the final performance of the net depend heavily on exactly when training is stopped.

**Boolean Classification Networks**

The way networks have been used here, having a single PCN recognize all the poses at a node, requires that a unique network be trained for every node which branches on a different set of poses. This can present a logistical problem in maintaining and training many different PCNs. The problem has been addressed here by maintaining a library of training images so that nets can be trained ahead of time on any combination of poses, then loaded as required. An alternative approach would be to use one network for each pose the system can recognize which computes a Boolean predicate on that pose. In other words, each pose would have one network that determined if the input image contained that pose or not. The output of these networks could then be combined to meet the needs.
of a node branching on pose links.

To explore this possibility, a network was trained to perform the Boolean predicate for each of the three poses discussed here. The networks used were identical except that they have two output units, one corresponding to the presence of the target pose, one corresponding to its absence. The nets were trained using the same image sets, training schedules and image modification parameters.

While the Boolean nets were able to learn the training set to near 100% accuracy, they were not able to generalize as well, only getting 80-85% accuracy on the test set, and performing even worse during interactive testing. On this basis a decision was made to stay with the full classification networks.

It may be possible to improve this performance of the Boolean nets by tuning the training set and perhaps the training schedule or network architecture for this task, but this has not been attempted here. Judging by other similar work [RBK96] one possible improvement would be to greatly increase the number of non-target training examples. These could be generated randomly or by a pseudo random technique such as used in [RBK96].

4.4 The System as a Whole

4.4.1 Speed

Using a relatively slow frame-grabber/DSP board set (see Section 3.7.1 for details) installed in a 100MHz 486 PC, the system can segment and track a hand at 7-8 Hz. This speed is acceptable in a prototype, but experience with the system suggests it is somewhat to slow for a practical interface. Subjectively, tracking feels good to the user because the smoothing algorithm ensures that the cursor catches up to the hand very quickly on large displacements, and moves immediately on small displacements, so the user does not notice any significant lag. Motion feature extraction does suffer from the slow tracking rate, however. As the user begins to get comfortable with gestural interaction, and so begins to move more quickly, extraction of motion features becomes less reliable, and the system begins to "misbehave". An experienced user can move at a smooth comfortable pace, but must perform important movements deliberately and without rushing in order for the system to understand his commands reliably. See Section 5.1.6 for a discussion of the causes and effects of a slow tracking rate.
Pose recognition operates at about 2 Hz. This is slower than tracking principally due to the additional image processing needed to accurately segment the hand and convert it to gray, as well as the handshaking needed to transfer the pose image up to the host. Propagating an image through the classification network on the host is not a significant contributor to the cycle time. Images can be applied to the net inputs, propagated through the net and their classifications retrieved at 65 Hz. This implies that given faster image processing, the networks could support higher resolution pose images, which may help improve pose recognition accuracy (but would, of course, increase network training times).

The relatively long time needed for pose recognition means that the user often has to pause briefly when a pose is presented. Little or no pause is needed for window selection, where the hand rises up to point at a window and immediately returns to the keyboard, since the system captures an image at the top of the hand's trajectory, then interprets the pose once it has verified the hand is indeed falling. So long as the hand paused long enough near the top of its flight for the system to capture a good image, the gesture is reliably recognized.

For some actions, however, the pose must be held till the system responds so that a subsequent action can be taken, for example when selecting an item from a menu the user must wait for the menu to be painted. Here the pause is much more distracting as the full time the user must hold a pose can reach one second. This includes the time needed for the system to recognize a pose classification is needed, the time for the DSP to crop and segment the previously saved image and pass it to the host, then the time for the host to classify the pose and respond with a visible action. A brief pause fits with the characteristics of natural gesticulation, as people often briefly pause to present a pose to the observer; however, the length of the pause needed here is longer than optimal. Users commented that the length of the pause was annoying but not a major problem, but we observed that it tended to disturb the rhythm of the gesture, making motions more awkward and so harder to interpret.

### 4.4.2 Task Performance

In an attempt to quantify the overall performance of the system, users were asked to perform a set of tasks on typical window configurations. A test set consisted of 50 to 100 repetitions of each of 4 tasks — selecting a window, moving a window, resizing a window and bringing up a window's menu (as opposed to the global system menu) and
selecting an item from it. The initial configuration of the screen consisted of 3 windows, each about 5” wide by 4” high, placed so they overlapped each other about half their area. The user was asked to perform the task on each window sequentially. Due to the slow tracking and pose recognition speeds, we did not feel it useful to time these actions or compare their speed to that of doing the same operation via mouse.

The total results for two experienced users are shown in Table 7. The qualified success rate indicates how many times the action was performed essentially correctly, but there was a “minor” error, such as the window ending up in the wrong position. The pure success rate indicates how often the action was performed exactly as the user intended. The minor errors nearly always have a common cause — trouble detecting where the retraction movement began, which in turn is primarily due to the relatively slow tracking speed rather than fundamental problems with the approach. The interested reader is referred to the discussions of sampling frequency in Section 5.1.6 and retraction in Section 5.1.3.

**Selection**

Selection, the simplest task, was also the most reliable. Only one core cycle is needed, meaning one pose classification and no possibility of the synchronization errors that can occur between the two phases of a more complex gesture. A minor error, selecting the wrong window, occurred when the system missed the top of the hand's trajectory due to slow tracking or sampling noise at that point. Nearly all the major errors occurred when the pointing pose was misclassified, usually as a palm pose.

The causes of pose misclassification were either timing errors or misclassification of a good image by the network. In the case of timing errors the pose image had been captured some time after the top of the trajectory, when it had started to relax from the pointing pose. Timing errors will be discussed in more detail shortly. When a good hand image was misclassified, the majority of the time it was an image taken near the top of

<table>
<thead>
<tr>
<th>Task</th>
<th>Steps</th>
<th>Trials</th>
<th>Qualified Success Rate</th>
<th>Pure Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>1</td>
<td>217</td>
<td>94%</td>
<td>89%</td>
</tr>
<tr>
<td>Move Window</td>
<td>2</td>
<td>206</td>
<td>85%</td>
<td>81%</td>
</tr>
<tr>
<td>Resize Window</td>
<td>2</td>
<td>238</td>
<td>87%</td>
<td>84%</td>
</tr>
<tr>
<td>Select Menu Item</td>
<td>2</td>
<td>99</td>
<td>82%</td>
<td>63%</td>
</tr>
</tbody>
</table>

Table 7: Results of system task testing.
the screen where the extended finger had been cropped off, making classification more difficult.

**Moving and Resizing**

Moving a window is a two-step command where first the user selects the window to move, then pulls their hand briefly back and points at it a second time to indicate move, then moves their hand to where they want the window to be. Moves were performed successfully 85% of the time. Of the successful moves, 4% of the time the window was placed in a position significantly different from where the user wanted, almost invariably because the system would follow the retracting hand for some distance before realizing a retraction was occurring.

The resize task is very similar to the move task, with the difference being in the second pose of the gesture. This provides a good check of our results, since performance numbers for resize, as well as the cause of the failures, are very similar to those for a move.

**Menus**

Bringing up a menu and selecting an item from it is the least reliable task. 82% of the time the system was able to correctly bring up a menu and allow the user to select an item from it. Unfortunately only 63% of the time did the system execute the item the user was trying to select. As with the other tasks, this was most often due to the difficulty of identifying exactly where a retraction movement begins. Since this is the most spatially exacting task, performance was more severely affected. The small size of menu items implies that even a tiny error in locating the start of a retraction will select the wrong item. A second contributing factor was that the menu items were small enough that increased difficulty of selecting small items, as discussed in section 4.2.2, becomes a factor. This aspect shows up in the form of user complaints about the difficulty of highlighting the desired menu item before they tried to select it with a retraction. This is in spite of the fact that the menu items had already been made much larger than they are in a typical mouse-based GUI. For a discussion of the trouble with menus and how they can be made more suited to gesture see Section 5.2.1.

**Types of Errors**

Table 8 shows the errors broken down by category. The numbers indicate the percentage of the total number of errors, major and minor combined. Retraction errors
indicate that the location of the beginning of the retraction movement was missed. Misclassification occurred when the pose recognition network classified a good quality hand image incorrectly. When a corrupted hand image caused a pose misclassification it was deemed a segmentation error. Tracking errors indicate that the primary cause of the error was that hand location was mapped to screen coordinates incorrectly.

Timing errors require more explanation. There are at least three causes of timing errors: hesitation by the operating system, race conditions in the transition network, and user movements faster than the system could successfully track. MS Windows seems to hesitate occasionally, perhaps as it accesses a swap file or performs some other long internal operation. During these periods, lasting from 0.5 to 2 seconds or so, interrupts are not processed. When the gesture system next gets a time slice, it appears that the user's hand has jumped. There are also certain paths through the transition network that are very dependent on timing. These can occasionally cause the system to take an image of the user's hand while it is in flight, resulting in a blurred image or partially formed pose, and so result in a misclassification. A careful examination of the transition network will be needed to excise such situations. Finally, if the user doesn't stop long enough for the system to detect the pause and capture an image for pose recognition, the hand will again be captured in flight resulting in a poor image. Because of the modest speed of this prototype, best recognition occurs if the user moves at a comfortable but unhurried pace. As the user becomes more comfortable with the system it is easy to begin to move too fast, and errors begin to occur.

One lesson from this testing, which does not show up in the performance numbers, is to emphasize the steep learning curve. When first working with the system, a user's actions are generally so hesitant that the error rate is very high. As a result, these numbers reflect the results of only two users who have used the system extensively. The steep learning curve and possible solutions for it are discussed in Section 5.2.1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Retract</th>
<th>Misclass</th>
<th>Segment</th>
<th>Tracking</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>18%</td>
<td>27%</td>
<td>2%</td>
<td>17%</td>
<td>36%</td>
</tr>
<tr>
<td>Move</td>
<td>21%</td>
<td>42%</td>
<td>3%</td>
<td>14%</td>
<td>20%</td>
</tr>
<tr>
<td>Resize</td>
<td>19%</td>
<td>37%</td>
<td>3%</td>
<td>15%</td>
<td>26%</td>
</tr>
<tr>
<td>Menu</td>
<td>51%</td>
<td>25%</td>
<td>&lt;1%</td>
<td>6%</td>
<td>17%</td>
</tr>
</tbody>
</table>

Table 8: Percentage of total errors by category.
4.4.3 User Comments

After they had a chance to use the system, most of the dozen or so people who have used the system were asked for their comments on its performance and usability. These people were nearly all computer professionals, some from the research community, and others from business. All had years of experience as users of traditional GUIs. This section summarizes what was said.

The aspect of the system receiving the best reaction was the ease of object selection. These comments were nearly always followed by a list of complaints with a mouse, from dirt on the ball making it erratic to loosing it under a pile of papers. One user was having trouble getting the system to consistently recognize her commands because of a tentative gesticulation style and hand poses significantly different in shape than those used to train the pose classification networks. Even still, after the session she commented, “If this behaved, it would be a lot easier to use than a mouse”. We went on to discuss her frustrations with a mouse, even after using one for years. Her chief complaints were the need to look away from the screen to find it, the difficulty mapping mouse movements to cursor actions, and the uncertainty when to double or single click (she often switched between Windows, OS/2 and AIX, each with different conventions). In general, the attractions of free-hand pointing seemed to be both the ability to do a simple operation without the overhead of hunting for a device first, and the directness of pointing rather than indirectness of mousing. The difficulty of selecting small objects appeared in complaints about the difficulty of selecting objects when they had only a small piece showing.

Moving a window by pointing at it a second time was also judged to be very natural, although the pause required in the middle of the gesture was irritating. Resizing windows was judged to be frustrating because of the rigid algorithm of only being able to pull around the lower right corner of the window. This often requires the user to move the window before resizing it.

Menus were not popular. The length of the interaction combined with the precision needed to select an item was judged to be tiring. Of course the irritation of then selecting the wrong item a third of the time magnified the problem.

Finally, the limitations of this prototype received comments. Many users expressed the desire to be able to select a wider range of actions using pose, but at the same time many had trouble remembering which pose corresponded to which action. This suggests the need for some kind of memory aids to help the user remember the pose to action
mapping. The inability to select sub-windows or icons was also cited.