

## 5 Discussion

This work leaves little doubt that the technical challenges of visually recognizing gesture in a practical domain can be overcome. The system as implemented is capable of recognizing hand gestures in a realistic setting in real time. It is very likely that with some engineering it could become a reliable add-on to current window systems. It would be easy to extend the system's capabilities so gesture could take on more of the interface duties. Solutions to remaining problems of reliability and accuracy appear to be relatively straight forward. Practical issues, such as easy calibration, present no theoretical hurdles. Section 5.1 discusses this work as a hand gesture recognition system. It examines what contributed to the success of the system at this level, and what holds it back from better performance.

This work also demonstrates the potential of hand gesture as an input device. After an initial learning curve, an experienced user can manipulate objects on the screen with speed and comfort comparable to other popular devices. The ability to interact directly with on-screen objects seems to be more comfortable to some users than the indirect pointing used in a mouse or joystick. The way that gesture has been used, however, is not optimal. Simply dropping it into an interface optimized for a mouse has been an interesting experiment, but has not improved usability significantly. Section 5.2 will discuss what has been learned about using hand gesture to interact with computers, and how an interface might be designed to make better use of their potential.

## 5.1 Vision Systems for Hand Gesture Recognition

This section will discuss the gesture recognition techniques used in this work. It first examines the various components of the system individually, to determine how well the approach taken to each aspect of the problem worked, and what might be done to make it work better. At the end of the section is a discussion of some general issues relevant to practical hand gesture recognition systems.

### 5.1.1 Segmentation

This work has shown that segmenting skin using low level clues, such as color, can work very well in an indoor office environment. The relatively unique color of human skin and the consistent good lighting combine to make good results possible.

Even in a relatively stable office environment, however, there is enough variation that the system must be able to be adjusted for the current conditions. The intended user is not going to want to fiddle around with extensive calibration on a regular basis, so fast and easy training is important. Since the training data will inevitably have errors, some approach similar to the color predicate training algorithm used here is necessary to get good results from a color-based segmentation algorithm.

Bottom-up segmentation algorithms like this will inevitably be noisy. While later processing can be designed to smooth it out or work in spite of its presence, it can still cause problems. Witness how the Type 1 noise (random jitter) described in Section 3.4.3 increased selection time for small objects, even after smoothing, and how the Type 3 noise (intermediate jumps) caused major problems for all components of the system.

There are several approaches to reducing noise. Higher resolution tracking and additional local image processing, such as morphology, should help a great deal. These approaches require significant processor power, however, and so introduce the trade-off between speed and accuracy discussed in Section 4.2.2. For a commercial system, that trade-off must be examined closely and optimized. Section 5.1.6 will discuss the speed requirements of a gesture-recognition system in more detail.

Another way to get more accurate segmentation is to incorporate a 2D hand model with constraints to limit the frame-to-frame variation of the segmented region. This will be particularly effective with the troublesome Type 3 noise. Unfortunately, even with advances in processor speed, model-based segmentation is unlikely to be fast enough to support the response times needed for interactive systems in the near future.

Finally, any practical system must be able to deal with the user's bare arm. Here again, domain knowledge is the best tool. In applications where the user is at a workstation, as opposed to moving freely in space, the imaging conditions are relatively stable so that the arm appears in a predictable place in the image with predictable shape parameters. In this work, the arm was characterized by a uniform and relatively constant size from near the bottom of the image up to the base of the hand, which is always wider. With a little thought, and some additional compute power, these characteristics could be used to excise it from the segmented image.

### **5.1.2 Tracking**

Using the centroid of the entire segmented hand blob to identify where the user is pointing works surprisingly well. The low resolution tracking window and simple image-to-screen mapping function used here are sufficient for comfortable interaction with icon-sized and larger screen objects.

This approach has the advantage of simplicity and speed over other approaches, such as tracking only the pointing finger (ignoring the body of the hand), or using 3D models to determine where the finger is pointing. The disadvantage with respect to those approaches, is that it relies more heavily on a consistent relationship between the appearance of the hand and where it is pointing. When that relationship changes, recalibration is necessary. Although calibration is relatively painless, it is not something the user is going to want to do more than once or twice a day.

This places limits on the applicability of the approach. For example, the user must interact with the system from a consistent position. It is not well suited to an environment where the user may access the system from a sitting position one time, and while standing another. Another limitation arises from the variability in pointing hand shape between users. The current approach is best suited to a situation where one person uses a system for extended periods of time, but is less appropriate when different users need to perform short interactions, unless it is practical to calibrate before each use.

For free-hand pointing to be able to support a realistic interface, the ability if the user to interact with small objects must be improved. Two changes will address the majority of the problems encountered here. First, reducing the amplitude of the random noise in the hand location, as discussed in Section 4.2.2 and 5.1.1, will make object selection faster and less demanding. Second, the polynomial warp from image space to screen space described in Section 3.3.1 will make the relative position of the cursor to the hand

more stable, reducing local irregularities and so making it easier for the user to predict where the cursor will fall.

Tracking noise can also be addressed at higher levels of the system. For example, if nothing is located where the user seems to be pointing when they request a manipulation function, the current system assumes they are referring to the nearest valid object. This works well when the screen is relatively unpopulated with objects, but it does not help, for example, when only a small portion of a window is sticking out from behind other windows. High-level heuristics, such as this, can be used to reduce the affect of tracking noise on user interactions, but by their nature they will tend to be domain specific. A similar use of heuristics is discussed in the next section for reducing the effects of tracking noise on motion feature extraction.

In this system, and likely in most systems using indirect sensing, it can be difficult to have uniformly good behavior everywhere in the workspace. The design of a vision-based gesture recognition system should take this into account by designing in a “sweet spot” that corresponds to where the user would naturally do most of their detailed work. Resolution can be allowed to fall off in other areas. The interface software can take this behavior pattern into account by using interaction modes that do not require high resolution where tracking behavior is poor.

### **5.1.3 Motion Feature Extraction**

Motion feature recognition is the weakest link of this implementation. False positive features are common and the false negative rate is higher than ideal. This forces the motion interpretation step to deal with a very noisy motion feature stream, which in turn makes designing a robust interaction language a long process involving a lot of trial and error.

This problem is caused in part by the slow tracking rate. The effects of tracking rate on motion feature extraction will be discussed at length in Section 5.1.6. Another contributing factor is that the way complex motion features, like the Comma, have been implemented, they are very sensitive to timing. If the user performs a movement faster or slower than expected, the recognition operator is likely to miss the feature. This suggests that more robust recognition methods are needed. Both dynamic time warping algorithms like those used in speech recognition [SC80] and Hidden Markov Models [RJ86] should be investigated. It may also be fruitful to explore work in general path understanding such as [Bu83] or [Li91]. I should emphasize I am referring to applying these techniques

to recognizing individual motion features, not to the entire motion/pose sequence of a gesture. HMMs may not be the best approach for recognition of complete gestures for the reasons discussed in Section 3.6.1.

Another cause of poor motion feature extraction is caused by a flaw in the implementation which the alert reader may have picked up. The smoothing algorithm uses a model including a velocity vector to compute each new cursor location, but only the final location, not the velocity, is recorded for use by the motion feature extraction routines. When the velocity of the cursor is needed to detect a motion feature, the extraction operators must examine the sequence of positions to re-compute a coarse velocity vector. The detection of several motion features could be made easier and potentially more accurate if the smoothing algorithm returned a more complete description of the cursor state including both position and velocity, especially since with the current path smoothing algorithm, velocity encodes information about the relative position of the cursor and the hand which is not immediately obvious from the sequence of locations. Unfortunately, this situation arose unnoticed as the system evolved, and was not detected till it was too late to correct within the time frame of this work.

Two motion features, Pause and HandPresent (and its derivatives HandGone and HandAppear), are likely to be important in any application, but have proven difficult to accurately detect.

To detect a Pause it is not sufficient to simply use a threshold on the movement over one cycle. Type 1 noise causes even a motionless hand to elicit some degree of “motion” in the cursor. The smoothing algorithm eliminates the highest frequencies and amplitudes, leaving a slower wandering movement. Unfortunately fine positioning often uses movements of the same amplitude. It is not possible to simply adjust the parameters of a knowledge poor smoothing algorithm as this damps out the high frequencies needed for good visual tracking and by other feature extraction operators. The approach taken here has been to only detect a Pause if the total movement over two cycles is below a threshold, to look for some amount of consistency in the motion vectors, but to still provide the user a good response time. This has proven to be usable, but tends to produce too many false positives.

People naturally move differently when trying to finely position something. For example they tend to use slow, deliberate motions, and they tend to use a move-wait-move pattern. This suggests two approaches to improve Pause detection. First rather than looking for changes less than some absolute amplitude, the Pause operator could

examine the direction of the motion vectors. The longer the cycle-to-cycle motion vectors are close in direction, regardless of magnitude, the more likely they are to indicate purposeful motion, i.e. fine positioning. This is equivalent to looking for low frequency motion components regardless of amplitude. Second the paths could be examined for evidence of the move-wait-move motion pattern of the proper amplitude to suggest the user is fine-positioning. Either approach is likely to require faster tracking rates to ensure good user response times.

The second problematic “motion feature” is detecting when the hand is present in the image. This can become a serious problem because the disappearance of the hand is critical to detecting a retraction for the reasons discussed in Section 5.1.6, and retraction is an essential feature in all of the gestural interaction languages that have been explored. Detecting the presence of the hand is currently based only on a size threshold on the segmented region. While this gives good performance most of the time, occasionally it becomes unreliable. Some people tend to gesture with their hand far from the screen, making its image relatively small, and so making it difficult to set a size threshold that detects the hand but not the face or other image noise. Finding an appropriate threshold can also be a problem in very noisy environments.

A more robust algorithm would depend on not only the size of the hand, but its path. It is very unusual, for example, for the user to pull their hand straight back from the screen during a retraction. Much more typical is to drop it toward the keyboard. If the system detects the hand pulling straight back before a disappearance, it should wait for some number of cycles to see if the hand was simply pulling back as part of a Comma or some other motion. Another example is that it would be very likely for a hand to be traveling down toward the bottom of the screen, then suddenly bounce up to the center of the screen at a smaller size. This is exactly the behavior that the system sees if the threshold is too low, however, as the tracking algorithm tracks the hand till it disappears, then switches to finding the face. Clearly, incorporating heuristics to detect these types of situations would make detection much more robust.

These examples lead to a general design principal which is to use multiple sources of redundant data to recognize motion features whenever possible. Basing feature detection on a single datum makes it unreliable. One of the more reliable features to be detected was the Comma, which required five conditions to be met for detection. Unfortunately, that approach often requires a faster tracking rate, so that information can be gathered across some number of cycles before a feature detection is triggered.

Of course the trade off with this approach is that any one of those events is not detected, the feature as a whole is not detected, leading to a higher false negative rate. Our experience indicates, however, that this behavior is preferable to detecting a feature too often. When a feature is falsely detected there is a fair chance it will trigger an unintended action to be performed. Thus false positive features lead to the system doing something the user does not expect, and they get confused as to what changed and how to recover. A false negative, on the other hand, generally leads to the system failing to perform an action. At worst, this will cause the user to perform the action again, giving a higher frustration factor but a safer failure mode.

Finally, when this work was begun, it seemed that recognition the gross movement of the hand was all that was needed in this domain. By the end of the work, however, it became clear that higher level descriptors of the motion were needed, such as the Comma and Retraction. As a result, detection of these complex motions has been pushed into the interaction language itself, where they must be done based on the coarse motion predicates. For example the language now looks for a retraction in a sequence of coarse motion vectors in the FollowDown node. This approach is cumbersome and puts an undo burden on the language programmer. The set of motion features extracted atomically must be expanded to include these more elaborate features.

#### **5.1.4 Pose Recognition**

Template matching for pose recognition is well suited to the domain. It is trainable, which is important to compensate for the variation inherent in such a domain. It is appearance based, which is arguably preferable to model-based matching (see Section 3.5.1). The particular approach taken here, using a neural net, produces good results with an acceptable amount of training in terms of both time and number of training images. It is easily fast enough during recognition to meet the needs of a real-time application.

The preprocessing steps, which remove as much of the variation in the image as possible, are important to obtaining good results. Minor imperfections are not as important as consistency of the result, as any learning algorithm will be able to compensate for a reasonable amount of noise. Essentially, the more consistent the result of preprocessing, the smaller the training set can be and the simpler the recognition net (or equivalent) can be as less variation need be taken into account.

While the pose recognition performance achieved here is very good for a prototype, improvements are needed to support a truly robust interface. Several avenues for

improvement are suggested by this work.

The biggest single factor in improving pose recognition performance during development was the size of the training set. Performance could almost certainly be improved further by expanding the training set even more. The current set of 40 or so images of each pose only contains two or three images of each pose from each orientation, and all images are from a single user. This is barely enough to represent the range of variation in appearance. Ideally it should be expanded to include base images from a variety of users. The current training images are modified in scale, rotation and translation, but the intensity values are not modified. The possibility of adding perspective to the modifications to more realistically model the expected variation in the input images was discussed in Section 3.5.1. It is also reasonable to expect that if a noise model were used to modify the training images as well, say including facsimiles of the user's head or simulating a poor segmentation, that performance could be improved.

The effect of different network architectures should be explored further. During development, the number of hidden nodes was increased from 5 to 20. As a result, recognition performance went up considerably, at the cost of somewhat greater training times. A weight analysis (Section 4.3.4) before and after the change showed that the 20 node network was detecting more distinct, well-formed features in the hidden units. This suggests that the hidden units of the 5 node network were recognizing multiple features, causing potential interference. At a minimum, the relationship between performance and the number of hidden nodes should be explored more completely.

The approach taken here of having a unique pose classification network (PCN) trained for each node in the transition network which branches on pose links may well prove cumbersome in a more complex domain. Therefore it would be interesting to explore alternative ways of using the network. For example, it may be possible to use a single PCN for all nodes requiring a pose classification in a language. If there were no links leaving a particular node corresponding to the pose as which the image was classified, an error could be flagged. This approach may give more chances to catch user errors. It may also be possible to train binary networks, as was discussed in Section 4.3.5. These tests and other work in the area [RBK96] indicate that if a suitable negative training set can be created, nets may be able to learn a Boolean classification function. Sets of these nets could then be combined as needed.

While template matching is good at differentiating poses using global rather than local information, there may be applications where distinctive local features of the pose are

easy to find by some other means. For example, an application may want to allow the user to hold a pencil while pointing to get more accurate spatial localization. The easiest way to determine if a pencil is present in the image may be to detect the triangular head or straight shaft by convolving the intensity image with a mask. It would be relatively straight forward to include such a step in the pose recognition routines before the networks are asked to classify the image. Combining traditional feature detection algorithms with network-based template matching can allow the pose recognition step to make use of whatever features are best suited to the problem at hand.

### **5.1.5 Language Representation**

Using a transition network to describe the interaction language worked out very well. It proved flexible and powerful enough to handle any variations to the interaction language we cared to try. It was easy to change and clear to understand.

The major flaw with this approach was that while it was easy to get something working, and to make modifications, it was difficult to implement the desired interaction in a robust way. It was too easy to introduce timing dependencies, often leading to timing errors (Section 4.4.2). It took a great deal of thought and experimentation to make the interaction respond well in the face of noise in the extracted feature set. Many of these problems will be solved by correcting for timing problems and false positives in the feature extraction routines, but it may be possible to develop language writing tools and conventions that are robust to common problems. This is where a careful study of work in User Interface Management Systems could contribute. See Section 3.6.1 for references and some more thoughts on the subject.

The interaction language itself will be discussed in Section 5.2.1.

### **5.1.6 General Considerations**

#### **System Speed**

The observation that the current tracking rate of 7-8 Hz is somewhat too slow for good usability fits well with the observations of others (e.g. [CNM87]). This section will examine the effects of tracking rate on this system in more detail, and suggest ways to improve it.

An inadequate tracking rate causes two types of problems with this system. The first is that the cursor will tend to lag noticeably behind the hand, making it difficult for the

user to position the cursor where they want it. Due to the non-linear response of, the smoothing algorithm, this effect is only noticeable below about 5 Hz, so at the current tracking rates the user's perception is that the cursor tracks the hand well.

The second problem with a slow tracking rate is a reduced reliability of motion feature extraction. At the current tracking rates, this can become a problem if the user moves too quickly. Specifically, slow tracking rates cause problems localizing motion features, as well as both false positive and false negative feature detection. Problems with feature localization are obvious. Consider trying to determine the exact point where a motion reverses. One sample will be taken on the way to the reversal point and one on the way back. As the sampling rate decreases the average distance of those samples from the reversal point will increase, making precise localization more difficult.

The causes of false positive and negative feature detection are more subtle. The underlying cause is that there are relatively few samples taken which lie on a feature. For example the most complex motion feature, the Comma, takes place in about 1/2 second, so that only 3-4 samples are taken while it is occurring. This has two implications. First, with fewer samples on a feature, the relative importance of each sample is much greater. This means that if one of those samples is corrupted by noise there is a much higher probability that the feature will not be recognized. This, of course, leads to false negative features. In the case of the Comma, it is currently modeled by four samples, call them 1-4. Looking just at the Y coordinate for this discussion, Y must decrease between 1 and 2, its behavior between 2 and 3 is ignored, then it must be increasing between samples 3 and 4. If sample 3 is corrupted by noise so that Y is not seen to be increasing between 3 and 4, there is no way to recognize it. If the sampling rate were doubled, so that there were twice as many samples were taken during a Comma it would be possible for an intelligent operator to ignore the noise at any one sample, and so still recognize the feature.

The second implication is that there is not enough redundant information to verify that a feature is indeed occurring. In this case one or two noisy samples can lead to false positive feature detection. An example of this problem occurs in detecting a retraction. At the current sampling rates, the hand can go from being paused at some screen location to being out of view in one cycle. From the point of view of the system the hand simply disappears. There is no choice but to assume the user quickly dropped their hand to the keyboard, and so to interpret this disappearance as a retraction. However it is possible that the user could have moved their hand to another portion of the screen, where local segmentation errors caused the size of the segmented region to drop below the minimum threshold for a cycle or two. Given a faster sampling rate, the detection algorithm would

be able to verify that the disappearance was indeed a retraction by checking that the hand was descending toward the keyboard before it disappeared, and in the absence of that verification, defer reporting a retraction. Unfortunately there are simply too few samples that occur during the feature for that type of analysis.

So clearly, too low a sampling frequency is a real problem for reliable motion feature extraction. Is it possible to determine what the minimum frequency should be? Using a simple experiment, where you move your unsupported hand back and forth as fast as possible, the fastest you will be able to comfortably achieve is about six Hz. This fits with the observations of others, for example in [Br95] Section 2.1, he observes that the highest frequencies of target movement a subject can track with a pointer seem to be around 5 Hz.

Considering the motion of the hand as a time varying signal, and tracking the hand as digitally sampling that signal, then information theory tells us that the sampling rate needs to be twice the maximum frequency present in the signal to accurately capture all the information present. This suggests that the minimum tracking speed for reliable hand gesture recognition should be about 10-12Hz. This seems to fit well with the observations made here, that the 7-8 Hz tracking rate allowed acceptable detection of motion features until the user started to become very comfortable with the system and so move quickly. Experience with slower tracking rates during system development indicates that below about 5 Hz a user must move artificially slowly for the reliability of motion feature extraction to be acceptable.

There is room in both the hardware and software for improvements in tracking speed. Currently, limitations on tracking speed come from the time needed to transfer images between the frame grabber and the DSP, the processing speed of the DSP, the efficiency of the image processing algorithms, the time needed to transfer results to the host, and the speed of the host itself.

Moving the system from a 66MHz to a 100MHz host improved tracking performance only very slightly, evidence that the host speed is not a serious bottleneck. The maximum rate images can be transferred into the DSP is about 15Hz, which does slow tracking considerably, especially since this can not be done concurrently with image processing on the DSP. Therefore, substantive gains in tracking speed will be had from a port to more modern image processing hardware. In addition to faster DSP clock speeds, this will eliminate the need to transfer a full-sized image between boards every cycle. Further gains can be made from special purpose hardware. Assists for HSI/RGB conversion,

resolution reduction and morphology would be beneficial.

There are also several ways the image processing algorithms can be optimized for tracking speed. The most obvious is to take advantage of the positional information extracted in one cycle to focus processing in the next (i.e. Region of Interest processing). For simplicity, this implementation simply processed the entire image each cycle. Additionally, many of the computations that were explicitly computed during smoothing, warping and image preprocessing could be converted to look-up tables.

### **Handling variation**

Humans are notoriously variable. Any system designed to interact with them must either limit that variation, as is done with current input devices, or be able to work in its presence. Since visual gesture recognition has no built in way to limit variation it must be able to work reliably in the face of a wide range of gesture styles. While the ability to learn this variation was designed into this system in several areas, there were two types of variation that were not anticipated and caused some problems.

The first was the seating position of the user. The system was intended to be used as the user was typing, so that the user would bring their hand up from the keyboard to manipulate an object on the screen and return to typing. What we found was that when we demonstrated the system to users, since they were not constrained by the keyboard (there was no need to type), they would lean way back in the chair or lean very close to the screen, but rarely sit up as they would when they were typing. In use this probably represents a realistic situation, as users often take a break from typing and relax somewhat when they have to manipulate the screen.

This variation caused problems in two ways. First the appearance of the hand pose would change as the angle from which the arm was approaching the screen changed. Secondly the mapping from the image of the hand to the point on the screen the user was trying to indicate would change to some degree. It should be possible to handle the first problem sufficiently by incorporating a wider variation of hand appearances into the pose recognition training set. The second problem is more of a problem. It may be possible to observe other parts of the user than just their hand to adjust the image-to-screen mapping parameters. For example, to use the size and location of the user's face to determine how they are sitting and so adjust the mapping parameters.

The other unexpected source of variation was the extreme variability demonstrated by new users. Both the shape of hand poses and the pace at which motions were executed

would change radically during the first few minutes with the system. They would form the poses differently each time, be hesitant about their motions, hold their hand close to the screen one time and much further away the next. Even when asked to be as consistent as possible, the details of their gestures varied a great deal. To compensate, again the pose recognition training set must include at least some of the awkward poses formed by an inexperienced user, but also the motion feature extraction routines and the transition network must be able to handle a wide range of variation in pace and timing of movements, as discussed earlier.

For a truly flexible interface these steps will not suffice. Current user interfaces are very constrained in the actions a user can perform. As we move toward interfaces that allow the user more freedom of expression, the interface must be able to handle a wider range of interaction styles. It is not enough to be able to train a system to recognize a particular style, because there is so much variation not only between users, but with a single user depending on his energy level, mental state, etc. Systems of the future must be able to dynamically adjust themselves to the current state (physical and otherwise) of the user.

### **The importance of context**

One of the things that makes a working system possible in a difficult domain such as this is a liberal application of domain specific knowledge. Clearly, any system designed to perform a real task must incorporate domain specific knowledge at some point, but it is possible to put off application of that knowledge till the very end, in other words use general purpose techniques in the low and middle levels to construct an internal representation that can then be examined by a model specific component to perform the task. The advantage of this approach is that hopefully the lower levels of the system can support a wide range of tasks and conditions because they incorporate as little domain specific information as possible.

Unfortunately, general techniques yield cumbersome solutions that have little chance of providing real time response and greatly increase the complexity for the system builder. If the goal is to have a working system, the constraints provided by the goals for the system itself can make the search for a solution much easier if the designer is willing to take advantage of them.

We have used domain specific knowledge to tune relatively simple image processing techniques, guide their application and combine their results in interesting ways. For example, because of the imaging conditions the system can assume that whenever it sees

a skin-tone blob of greater than some size it can assume it is a hand without checking its shape. Thus the lowest levels only need look for large skin-tone blobs.

This kind of domain knowledge is applied at every level from the lowest to the highest, and is made explicitly available to the system when appropriate. There are two distinct types of domain knowledge that we refer to as the static context and dynamic context. The static context describes how this particular task fits into the space of vision problems. It includes knowledge of the imaging conditions, the static and dynamic characteristics of the objects of interest (in this case hands), the characteristics of the remainder of the environment, and knowledge of the types of information that must be extracted from the image in order to complete the task. The dynamic context is concerned with how a particular instance of a task fits into the space of possible instances. In an interactive, conversational domain such as this it includes knowledge of what has been “said” up to this point, and what to expect next.

While static context has been applied to the design of all levels of the system, information about the dynamic context is generally only available at the highest level, providing the explicit knowledge it needs to keep track of the conversation with the user. The transition network allows the system to track the dynamic context, keeping track of features extracted by a suite of feature extraction operators over many frames. It not only tells the system when and how to respond to the user, it provides expectations that tell the system when to apply expensive feature extraction operators.

The dynamic context allows us to use relatively easy-to-extract image features which otherwise would be too unreliable to use. For example movement of a roughly skin colored, hand sized blob across the image by itself could just be someone's head as they walk through the lab. In the context of a sequence of other image events, however, it can reliably be assumed to be the user indicating where to put a window. There is no need to verify that the blob is indeed a hand with expensive high resolution processing. The dynamic context also provides a set of expectations that help guide application of the feature extraction operators, and so avoid unnecessary work. Only the events which potentially lead to another state need be extracted from the image sequence.

In spite of its reliance on domain knowledge, this architecture retains reasonable flexibility because of the modular nature of the feature extraction operators and the explicit representation of the interaction language. Minor adaptation, such as to account for different user preferences, is simply a matter of changing the language. A wider range of tasks can be accommodated by adding additional feature extraction and action

operators.

## **5.2 Hand Gestures as an Interface Modality**

The discussion will now turn from the gesture recognition issues to focus on the user interface. This work began without a clear vision of how hand gesture should be used in an interface. Indeed, one of the goals was to implement a flexible gesture recognition system, in order to be able to experiment with various different interaction styles. Unfortunately, only a few alternatives were able to be explored within the time frame of this thesis.

The results so far demonstrate that hand gestures can reliably perform a range of tasks in a user interface. At the same time it is clear that the way in which gesture has been used here is not optimal. This section will discuss what has been learned about using gesture to interact with a machine. Section 5.2.1 will examine some of the inherent characteristics of free-hand gesture that should be taken into account in the design of a gesture-based interface. Section 5.2.2 will consider how gesture can best perform some of the fundamental elements of a user interface. One of the important features of a gesture-based interface is the increased difficulty of learning the system. Section 5.2.3 will discuss the reasons for this, and present some suggestions for how it can be addressed. Finally Section 5.2.4 will present some specific suggestions for the design of a gesture-based workstation.

### **5.2.1 Characteristics of Free-Hand Gesture**

Except for a keyboard, most current input devices are essentially pointers with a button attached. They are designed to accurately identify a position using a relatively uninformative event (click, double-click, etc.). Hand gesture can emulate this interaction style reasonably well, but it also has capabilities that go far beyond simple pointing and clicking. Along with these additional capabilities, however, come some not-so-welcome idiosyncrasies. In order to make best use of gesture in an interface, its capabilities and problems must be well understood. This section will discuss some of the characteristics of hand gesture which have become obvious during the course of this work. The items are presented in no particular order.

## **Noise**

Visual recognition of free-hand gesture has more noise in the data-stream it produces than most other interface devices. Improvements in the recognition system can reduce the noise, but can not eliminate it. This is due, in part, to the need for non-contact sensing. It is difficult to track an object accurately in space, and to accurately classify its shape. Noise also arises from the inherent nature of hand gesture. Unsupported arms shake and hands do not reproduce poses identically every time.

As a result, any gesture interface must be able to deal with noise, not only in the position of the hand, but in any pose or motion features as well. The effect of noise on the ability to select screen objects was discussed in Section 4.2.2, but this is just one aspect of the problem. The interface itself must be designed to resist errors in the face of noisy input data. Moreover, since errors are more likely than they are with other input devices, it is very important that the system inform the user about what is going on, and allow them to easily stop or reverse any interaction. Methods to design an interface capable of good behavior in the presence of this type of noise is an area needing future work.

## **Spatial Positioning Characteristics**

Identifying large objects with free-hand pointing is intuitive and at least as fast as with other devices (see Section 4.2.2). Identifying small objects is more difficult. Much of that difficulty is currently due to noise in the detected hand position, but there is evidence that, even under ideal conditions, the difficulty of selecting objects increases faster with decreasing object size than it does with a mouse (see Figure 39). This characteristic clearly impacts the design of a gesture-based interface, if just in the size of the buttons and tools that must be manipulated.

Another characteristic of free-hand pointing that was observed in this work, but has not yet been objectively verified, is that vertical positioning is more difficult than horizontal positioning. This intuitively makes sense, as vertical positioning requires extending the hand away from the body with large muscle groups and fighting gravity, while horizontal positioning only requires rotating the shoulder or wrist. If true, this will have many subtle implications for the design of an interface. The menus designed for gesture in Section 5.2.2 are one example.

## **Temporal Positioning**

One task a mouse can perform very well is identifying a point in time. A user can

perform a mouse click at a very precise moment, as there is little movement involved and good tactile feedback as to when the event occurs. While there are temporal events that could be used with gesture, such a finger snap, a quick retraction, touching two fingers or touching an object like the screen, these all are likely to be less precise than a mouse click. Most require more movement of the hand or digits than a mouse click, most require more muscle energy. Unless the camera angle is just right, it can be difficult to determine exactly when many of them occur, and finally the complexity of visual feature detection is such that there is likely to be more lag before the event is reported than there is with a mouse click. For most interactions the difference in temporal precision is not likely to be a problem, there are no doubt some situations where it can be an issue.

### **Multiple Information Channels**

Hand gestures can easily deliver multiple pieces of information at one time. They can concurrently convey information via pose, position, and motion. These channels can carry separate kinds of information, or can complement one another.

To some extent, the same is true of other devices, e.g. a mouse can convey information via position, motion, and button status, but this has generally not been utilized. Interpretation of two dimensional movements has received only limited attention (e.g. [Bu83][Li91][Ru91]), and with a mouse, motion gestures are severely limited because it is not natural to lift the mouse off the table as you would lift off the paper at the end of a pen gesture. While there are a large number of possible button press combinations, their utility is limited by their similarity. Ignoring double-clicks, a three button mouse can have 7 different types of "click", which can be used in combination with keys, such as "Shift", "Alt" or "Cntl", to make a large number of variations. Unfortunately, there is little in the way of distinguishing characteristics between them, and so little to help the user remember when to use which. Perhaps as a result, only a few combinations are ever used, and mouse interaction is based almost exclusively on position combined with one or two different types of click.

With gesture, position is not currently as accurate as with a mouse, but both motion and pose are much richer communication mediums than their mouse counterparts. Since both motions and poses have a great deal of "semantic texture", it is often possible to assign them an interpretation which makes sense to the user. The meaning conveyed by different channels can then be combined to create a rich gesture language. Section 5.2.2 will give some examples.

## **Positional Independence**

Gesture can easily relay information independent of position. While this is possible with point-and-click input devices by ignoring the position of an event, the events alone (e.g. the click) are so information-poor that not much useful information can be conveyed. With hand gesture the channels of pose and motion are so much more expressive, so a broad range of positionally independent interactions is possible. An example of where this capability would be useful is when the display area is extremely large, say the size of a desk or wall. If commands are always tied to a specific location, great care will have to be taken to avoid the situation where the user will have to walk from one side of the display to another to perform a simple action. With the ability to convey a wider range of commands independent of position, the problem is more easily avoided.

## **Remembering Gestures**

Because hand gestures are more complex than current mouse-based interactions, they are inherently more difficult to remember. Even in the simple interactions described in this work, new users had trouble remembering the gesture sequence.

The problem is complicated by a lack of built-in memory aids. Since mouse-based interactions rely so heavily on position, the system designer can put an icon at the position associated with a mouse click, which helps the user remember what actions are triggered there. Other cues can be built in, such as displaying a help string when the mouse gets near such a control point. Because gesture can convey information independent of position, there is often no natural place for a memory aid.

This problem can be addressed in several ways. A cue-card could be hung on the side of the screen. The system could draw prompts on the screen when it detects the beginning of a gesture, helping the user remember what alternatives are possible in the current context. Most importantly, the interface can be designed in such a way to aid its memorization. This last approach was taken here, using a fixed two-step gesture syntax and a sentence-like structure. These techniques proved to be inadequate for easy memorization, and at the same time they seriously limited the flexibility of the interface. The design of a powerful and easy to remember interaction style is clearly an area that needs further work. Some suggestions appear in Section 5.2.2.

## **Three Dimensional Input**

Since hand gestures occur in space, rather than on a surface, positioning is inherently

three dimensional. This can obviously be an advantage when manipulating objects in 3D, but it may be possible to take advantage of it even in the context of an essentially two dimensional user interface. Current GUIs are actually not two dimensional, rather “2.5 dimensional”, in that objects can be stacked in front of one another. A simple way the third dimension of hand position could be used is to determine the stacking order when an object is moved. Say the user had one window in front of them, and wanted to reach over to drag another window close to refer to some information on it. As they drag the window, the location of the hand in the first two dimensions would determine where the window was placed, while the position in the third dimension would determine whether to put the new window on in front of or behind the current one when they overlap.

### **The Midas Touch**

Hands never turn off. Whenever they are visible, the system will attempt to interpret them. This is similar to what has been called the "Midas Touch" in work on touch screens, where the users' wrists, elbows, or even inadvertent finger presses cause unintended actions to be performed. The problem is potentially worse with vision-based gesture systems. With a touch screen, the user can be aware of when and where they are touching by tactile feedback, and so has some degree of control. With gesture, there is no tactile or visual cue that the system may be watching. It is not clear exactly where the borders of the workspace lie. Inadvertent actions can occur without the user ever noticing.

Fortunately there are several workable solutions for this problem. Maggioni in [Ma95] suggests illuminating the active region for gesture recognition with a cone of colored light, an approach that may work well in some applications. In this work, the system used the expectations provided by the interaction language to ignore most incidental gesticulation. If the user's hand appeared in the workspace, but did not follow the expected path of a known gesture, it was ignored. While this approach does not require any additional actions by the user, it can be a problem if a natural gesture corresponds to a command gesture. In this case, a common problem was that while talking to another person the author would often point to objects on the screen, which the system would interpret as a command to select or move that object. Care must be taken in the design of an interface to minimize this problem.

### **Constraints on Movement**

Certain types of interactions can often benefit from physical constraints. It is easier to

draw a straight line with a ruler than without. Most pointing devices have some kinds of constraints built in. A joy stick often has a mechanism to keep it centered in each dimension, making straight horizontal and vertical movements easy. A mouse rests on the table top, allowing the user to brace their hand and so move it very accurately. Gesture has very few such constraints, which will effect what things it can do well and what things it can not.

A related characteristic is that gesture is a direct pointing device [Sh92] with no stable frame of reference. When the cursor is positioned with a mouse, both the cursor and the mouse generally stay still afterwards, so subsequent movements can be made starting at that point. If the cursor later has to be moved a short distance to the right, the user can simply slide the mouse slightly to the right. With the style of gesture used here, each time the user goes to move the cursor, they start from scratch. This can be both an advantage and a disadvantage.

### **Range of Actions**

The range of poses, motions and combinations a hand can perform is very large. Many gestures correspond naturally to manipulation actions such as grasping, pushing, turning, etc. This helps the interface designer develop interaction styles which are a natural fit for many tasks. The range of gestures is not unlimited, however. It is constrained by the number of gestures which can be reliably differentiated, and by the number which can be remembered by the user. It is also limited in that many commands do not have a natural gestural analog. People are not good at remembering arbitrary mappings, such as between a gesture and a seemingly unrelated result, effectively limiting the number of useful gestures. So while the interface designer has a wider range of possibilities with gesture than other interface devices, there is still a need for other methods to elicit many actions, such as menus or stylized “tools”.

## **5.2.2 Designing an Interface for Hand Gestures**

The simplest way to use hand gesture in a user interface is to directly replace the mouse in a current GUI. Positioning can be done with free-hand pointing, and an equivalent to the click can be defined using either a movement of the pointing hand, or a button press with the opposite hand. This work has demonstrated that such an approach can probably be made sufficiently robust to support a practical user interface.

Using hand gestures in this way would have some advantages over current interface

techniques, as there would be no physical device to keep track of, break or maintain. On the other hand, this approach is not likely to produce a significant increase in usability, as it does not take into account the inherent strengths of gesture. Indeed there is likely to be a net decrease in usability, because current interfaces have evolved using a keyboard and mouse, and so are tuned in many ways to their idiosyncrasies. An obvious example is the proliferation of small active regions on the screen. A fundamental characteristic of a mouse is that positioning is precise and easy, but the set of “gestures” is information poor. A typical PC interface using a two-button mouse rarely uses more than the 5 gestures: Click (left, right, both), Double Click left (double clicking with the right or both buttons is rarely used), and Click&Drag (a.k.a. Rubber Banding). The first four of these are very similar and, other than selection, do not naturally correspond to many actions the user would like to perform. As a result, current GUIs rely on accurate selection of numerous small icons. Given the difficulty of selecting small objects with hand gesture, there is an obvious incompatibility.

In order for hand gestures to come into their own, the style of interaction with the machine must change. Some aspects of the interface can be adapted to gesture with simple changes, such as making buttons larger. As described in the previous section, however, hand gestures have some very fundamental differences from a mouse, so trying to incorporate hand gestures using small tweaks to a mouse-based GUI will at best produce a compromised interface. It can be argued that one reason the touch screen never became common in standard interfaces was precisely because it tried to be a plug-in replacement for a mouse when many of its inherent characteristics are so different. In areas where a new interface has been designed and the physical layout is altered to facilitate touch, as with the kiosks seen in various public places, touch screens have found a niche.

This work has attempted to take the first steps in the direction of adapting the interface for gesture, incorporating interaction styles which would not be possible with a mouse. The results show that usability was best when the interaction was designed specifically for gesture. For example, rather than manipulating windows with the mouse-based method of selecting a small control region, less spatially demanding combinations of motion and pose are used. Window selection and movement was judged to be very natural. For operations where the mouse-based interaction style was retained, the results were not as good. The style of pull-down menus, where the user extends their hand to the top of the screen to get a menu and retracts it straight back to select an item, is very comfortable with a mouse but based on the results shown in Section 4.5.2, it is not well

suited to hand gesture. Simple modifications, such as enlarged menu items and adjusting cursor behavior within the menu, were not sufficient to support a reliable interaction. Clearly, a different approach is needed.

The remainder of this section will examine some ways to design an interface to make the best use of hand gestures. The essence of most user interactions is selecting an object, selecting an action to perform on it, and supplying parameters to control the action. These elements must be combined in a rhythm which suits the interface device. Other elements, such as some kind of menu, will be needed in order to support a realistic interface. Ways in which each of these elements can be performed with gesture will be addressed in turn.

### **Object Selection**

In most current mouse-based GUIs, objects are selected by positioning the cursor within it, or some sub-region of it, then usually clicking. Once an object has been selected, it often becomes the default for subsequent action commands.

This approach can be used in a gesture-based system, and is essentially the approach that has been used in this thesis. Since the cursor always follows the hand, even as it retracts away from the selection, simply selecting the object under the cursor is not sufficient, but some analog for the click must be defined. This could be a keystroke with the non-pointing hand, or a movement with the pointing hand, such as extending the thumb. In this work it is a Retraction or Comma movements. Each of these “clicks” has associated problems that must be addressed to make them practical. A keystroke must be differentiated from regular typing. Any “click” movement of the pointing hand may have the effect of changing the cursor location slightly. The problems with detecting the starting point for a Retraction have been discussed elsewhere in this thesis (e.g. Section 5.1.6).

Positioning the cursor and "clicking" is an interaction with a wide range of potential applications. The natural meaning of a click is "do it", but "it" can be many things besides selection. Mouse-based systems handle this ambiguity with accurate positioning, small regions have different definitions of "it". Since accurate positioning is not well suited to hand gesture, this work relies primarily on the pose of the hand to refine "it", e.g. only a pointing pose elicits selection.

Using a pointing pose for selection is easy for the user to remember, but it is not the whole solution, as a pointing pose is a natural gesture for many different interpretations.

To differentiate between the interpretations requires yet another channel of information. In this work the context of a pointing pose within the gestural sentence helps determine its meaning. When a pointing pose occurs as the first pose of a command, the target object is selected. If it occurs in the second phrase, it implies a desire to move the selected object.

This example illustrates an important aspect of gesture interface design. The various components of a gesture — motion, pose and position — often have several different natural interpretations. Therefore it is important not to rely on any one component to determine the user's intent. Rather the system must examine these elements in combination. Other methods of overloading a pose with multiple meanings will be discussed later in this section.

A simple extension using gestures alone for object selection is to have the user hold a pointer to indicate selection. The user could pick up a simple stylus or pencil with a sharp point when they wanted to select an object. It would be a simple matter to examine the image in the area of the hand to see if it is holding a pointed object, the presence of such an object would indicate selection. This approach may allow more precise positioning during selection, depending on the physical layout, so could be useful when selecting small items, for example selecting text in a word processor. It would have the obvious disadvantage of requiring the user to pick up an object before selection, so is best suited for a domain where selection is relatively uncommon.

Clearly, there is no one right way object selection should be performed. All the alternatives discussed here have advantages and disadvantages. The correct choice should depend on usage, and a single system may well incorporate more than one selection mode. The same is true of the action selection techniques discussed next.

### **Action Selection**

In current GUIs, actions are most often called out by the position of the mouse during a click. Objects have numerous control points, indicated by icons, and the one containing the cursor determines the action performed. This approach could be easily duplicated with gesture. While the control points would have to be made larger and perhaps changed slightly (a scroll bar would be hard to control with gesture), but the approach would, on the whole, work. This is not the best way to design a gesture-based interface, however, as it does not make use of the inherent capabilities of gesture.

In addition to position, both the motion and pose of the hand can be used to elicit

actions. In simple domains, either the motion or the pose of the hand can be used alone. The most obvious way to select an action is to assign it a unique pose. This is a natural extension of direct manipulation, where the poses indicate physical analogs of real manipulation actions, such as grabbing an object to move it, but can easily be extended where the shape of the hand is not directly related to how the action would be physically performed. Hand motion can also be used to elicit action. Certain motion features map very well to some actions, and like pose, they can be extended to less intuitive mappings. A good example is making a circle with the hand to scroll. Circling in one direction would scroll a window forward and in the other direction would scroll it backwards.

As was mentioned in the discussion of object selection, using a single aspect of a gesture has inherent disadvantages. There are a limited number of possible poses and motions, each of which may have several different natural interpretations. By combining information from various different aspects of a gesture, the expressiveness of the gesture can be expanded. The way in which the elements should be combined depends on the application. Examples of interactions using various combinations of elements will now be considered.

**Pose and Position:** Combining pose and position is very natural for some operations. For example, it is possible to overload the grasping gesture with both the resize action it elicits in this work, and a move-object action, where the two interpretations are differentiated by position. If the grasp is performed near the center of a window, it would imply a desire to move it. If the grasp is performed near the corner of a window, it would imply a desire to resize the window by “grabbing” the corner and moving it. This approach requires that the objects or active regions to be selected with the hand position be sufficiently large that they can be reliably differentiated.

**Pose and Motion:** The pose of the hand and the motions that occur before and after that pose can be combined to elicit a wide range of interactions. A good example makes use of the natural semantics of Pause, Comma and Retraction. A Retraction naturally indicates that the user is done for now and returning to another task, say going back to typing. Pause indicates that the user has not yet finished with this command, so another phrase (such as indicating where the window should be moved) is to follow. A Comma indicates that they are done with the current command but want to do another. These intuitive meanings can be combined with poses to indicate a set of possible actions. In the interaction language shown in the Appendix, a pointing pose indicates selection if it is followed immediately by a retraction to the keyboard, but indicates a desire to move the window if, instead, the hand paused in a pointing position. An example that was not

implemented here combines a Grasp pose and the motions which follow it. A Grasp followed by a Pause indicates manual resizing, where the user controls the size of the window by subsequently moving their hand. A Grasp followed by a Retraction indicates a one-step resize, toggling a window between its largest and smallest sizes. A Grasp followed by a Comma indicates a one-step resizing, and the desire to perform another operation, such as moving the window.

Figure 46 shows an interaction language based on this approach. It implements the same range of actions as the language described in Section 3.6, but with the added benefit that actions on the current window can be performed in one step, without having to select it first.

**Multiple Pose:** It is possible to combine multiple poses by having the user move from one pose into another without moving their hand. The interpretation of the combined pose can be derived by combining the meaning of each pose individually. Imagine a gesture consisting of the Grasp pose closing into a Fist. That could indicate resizing to the smallest size, say an icon. A Grasp followed by a fully spread hand might indicate resizing to the maximum size, say full screen.

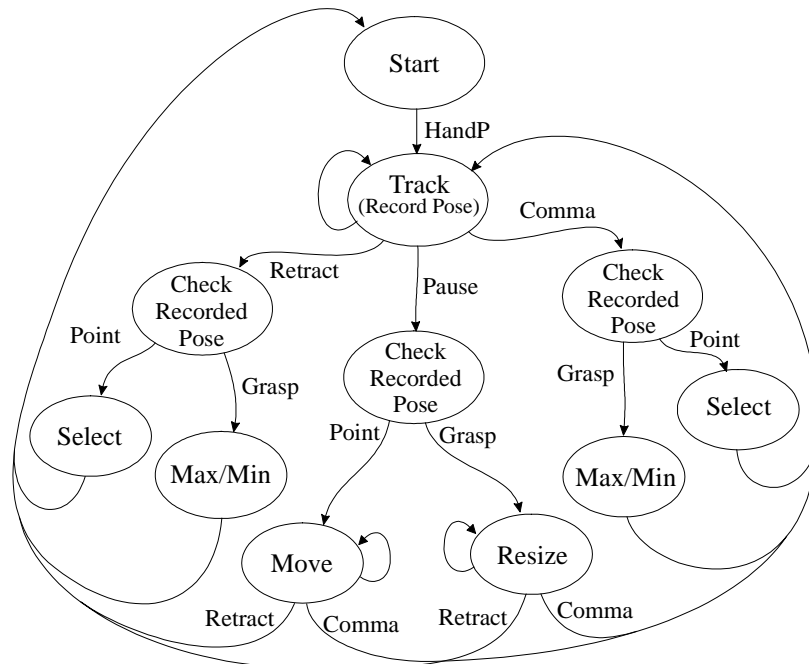


Figure 46: Alternate interaction language for the window control task, using the pose of the hand and the motion that occurs after it to signal an action.

## **Action Modifiers**

Another common element of graphical interactions is the ability to modify actions that have been requested. Where should the window be moved? How should the object be resized? In this work these modifiers are motions which are performed immediately after the pose signaling the action. A user points to a window to signal a desire to move it, then moves their hand where they want the window to be moved. This has worked very well, being intuitive and direct.

The modifiers used here tend to be analog, so naturally map to hand movements. It is possible to have discrete modifiers, however, and the gesture language could be extended to include them using the pose-pose interaction style described above. For example, it is possible to specify several discrete modifiers using the orientation of a pose. Imagine a mail system where a note can be selected and filed to one of several folders. If each folder is associated with a position around the periphery of the un-filed notes, then a user can move a note to a folder by pointing to it, then forming a fist with the thumb extending toward the target folder. Folder selection can be fine-tuned by rotating the fist. When the user retracts their hand, the note is filed in the folder last pointed to.

## **Rhythm of the interaction**

One area that must be addressed with gestural interactions, that is not an issue with current mouse-based systems, is the overall rhythm of the interaction. Mouse-based interactions use a few very simple rhythms. The most common is move-click. The most complex is the move-press-move-release of a click-and-drag operation. These rhythms are very well expressed by a sequence of events. Mouse-moves, mouse-button-down and mouse-button-up events can be combined in simple ways to express almost any mouse “gesture” which can be performed.

The rhythms of hand gestures can be much more complex. Hand gestures naturally have an underlying Prepare/Stroke/Retract cycle, as the hands are generally involved in some other activity, then come forward into the workspace to gesture and return. The stroke phase can be as simple as a brief Pause to display a pose, or it can be a longer sequence of motions and poses.

In this work several different rhythms were used. The simplest was the Prepare/Pose/Retract of a simple object selection. An item was selected from the system menu using Prepare/Pose/Pause/Select/Retract. Most object manipulations were performed using a fixed two stage rhythm of Prepare/Pose/Comma/Pose/Stroke/Retract,

where the first three steps selected the target object, and the last three controlled the manipulation of it. Various other permutations were possible depending on whether there was an action modifier, and other factors. Not only are the rhythms of hand gesture more complex than those of a mouse, but the individual elements are more complex than a simple event, so that interpretation does not lend itself easily to a simple event loop processing.

The fixed two stage rhythm used for most interactions has proven to be overly restrictive, preventing many natural or interesting gestures. A two-step command structure can be avoided by indicating the intended object concurrently with the action to be performed on it using different information channels within the gesture. The most obvious way to do this is to use the position of the hand as it requests an action. This is similar to the so-called “direct manipulation” interfaces often used in virtual reality interfaces. This approach can work well in some environments, in particular, when the representation of the environment is sufficiently large that the objects to be manipulated are unambiguously identified by the position of the hand. Consider a desk entirely covered with screen where life-sized representations of paper documents were the objects to be manipulated. The user would place their hand over a “page” and form a manipulation gesture, say “grasp” it to move it somewhere else. In the current setup, however, where the screen is relatively small, objects can be much smaller than the hand, and windows are allowed to overlap so that only small parts remain exposed. Here it would be difficult to get sufficient precision to select the desired target when using a large pose such as a spread hand.

Since the user may want to “say” more than one thing between a gesture’s initial Preparation phase and the final Retraction, some mechanism of separating the phrases must be found. It must be both quick and easy to execute, and reliable to recognize. The Comma motion, developed for this work, meets all these requirements. Here the user pulls their hand briefly back from the screen then returns it forward again. In this work a Comma played only a simple role, separating the selection and action

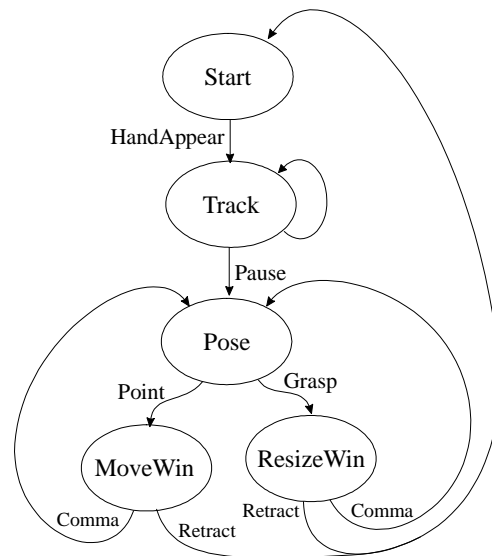


Figure 47: Interaction language allowing multiple actions, separated by a comma.

phrases of a gesture. To test it in a more complex sequence of interactions, a simple language was created which allowed the user to move and resize a window repeatedly, separating commands with Commas (Figure 47). The resulting interaction style proved to be very natural.

Because a Comma feels something like a partial retraction to the user, it is very natural to think of it as a minor break in the conversation. If a Retraction is analogous to the period at the end of a gestural sentence, a Comma is the comma between phrases of a single sentence. Pause can also be a useful separator, but it has a different natural meaning, more like a space between words in a sentence.

The following rhythm for a gestural interface addresses many of the points which have been raised:

Gesture -> Prepare <Stroke> Retract

Stroke -> [Phrase Comma]\* Phrase

Phrase -> [Pose|Motion Pause]\* Pose|Motion<sup>4</sup>

The syntax of the Phrase tokens can be further decomposed based on the discussions above to implement the specifics of the object selection, action selection and action modification in the interaction language.

This language is still limiting, however. Gestures are truly fluid entities, continuous combinations of motion and pose elements which may be better expressed as a dance than as an event stream. The elements overlap and modify one another just as words are modified by co-articulation effects. Some elements simply do not blend well together, and so are hard to perform. Other sequences flow off the fingertips easily. Rather than try to force gesture into the mold of token sequence, we should develop other ways of representing them, to allow them to reach their full expressiveness. An interesting area for future work would be to explore the representations and concepts used in choreography to see if any of them could be applied to gesture.

## **Hand Gesture Menus**

Even with the rich definition of gestures described above, it is not practical to expect that every possible action has a natural gesture to go along with it. What would be a natural gesture for query/replace, for example? Even if such a mapping was possible, in a

---

<sup>4</sup> Here [square brackets] represent an optional element, a vertical | bar indicates exclusive OR of the two elements, and [x]\* indicates 0 or more occurrences of x.

large system a user would have a very hard time remembering all the possibilities, especially for rare actions. Some form of menu is an obvious solution.

In the abstract, menus are well suited to gestural interaction. Navigating menus with direct pointing (as opposed to the indirect pointing of a mouse or joystick) is very natural. In [KMA86], Karat et.al. showed that using menus with direct pointing (in their case, a touch screen) was both faster and preferred by users over using either a mouse or keyboard. The implementation of menus used by most mouse-based systems, i.e. menus where items are arranged vertically above one another and selected by vertical motion, are not well suited to gesture. They require precise positioning in the vertical dimension and holding the hand up for relatively long periods of precise positioning, things hands are simply not good at.

For this work several minor changes were made to try to make mouse menus more suitable for gesture. The items were made thicker, and hysteresis and heavy smoothing were added. These changes helped up to a point, but taken beyond that they actually made menus more difficult to use. Large menus require large hand movements which are unnatural and uncomfortable. Too much smoothing makes the cursor lag behind the hand, making precise positioning difficult. Large amounts of hysteresis make it difficult to move between items. Unfortunately, even with these parameters set to their optimal values, menu selection was neither easy nor comfortable.

Clearly menus must be redesigned for use with gesture rather than simply adapted from the mouse. Is it possible to come up with a better way to choose between multiple options? The alternatives must be displayed for the user as a memory aid, but a vertical arrangement is a poor choice. With the standard screen configuration, vertical menus require the user to fight gravity with an unsupported hand while selecting items. The tension required for precise positioning tires the arm quickly. The hand becomes unsteady, requiring still more muscle tension. If the screen were instead laid flat, a vertical layout would still not be optimal, as pushing the arm toward and away from the body uses the large muscles of the upper arm and shoulder. These tend to be less precise and more tiring than movements of the smaller muscles of the lower arm. On the other hand, positioning the hand left or right is relatively easy by rotating the wrist or swinging the arm about the elbow. This suggests a horizontal layout for menu items.

If the items are displayed horizontally, rotational motion of the wrist and arm can be used to highlight the items and vertical motion used to execute them (Figure 48). This way the user can allow the vertical position to stay fixed isometrically, a much more relaxing posture than precise vertical positioning. Sub-menus would be called up by relatively large vertical movements, followed by another rotational selection phase.

The number of items visible on the screen would be much more limited than with vertical menus, but this should not be an issue, as the ideal number of menu items for even for vertical menus has been shown to be 8-9 [Ki84], which should fit well horizontally. If more menu items are really needed at a single level, they can be handled by allowing the menu to scroll left or right when the user reached the edge of the screen, though this would likely cause some decrease in performance [MP90].

Another menu design that would work well with gesture is “pie menus”. These are circular menus where each item gets a pie-shaped slice [Ca88]. A variation of pie menus, called “marking menus”, have shown good performance and several usability advantages on pen-based systems [KB93]. These would show similar advantages in a hand gesture interface.

### 5.2.3 Climbing the Learning Curve

As mentioned earlier, it takes a fair amount of experience with the system to be able to achieve consistently good performance. Initially most users are very frustrated getting the system to do even simple tasks. They can position the cursor reasonably well, but often have trouble with the rhythm of the hand movements needed to control objects. Over time performance improves as the user learns to provide inputs in such a way that the system recognizes it reliably. While this is true for traditional mouse systems as well — mouse actions such as the proper timing of a double click or the coordination of a

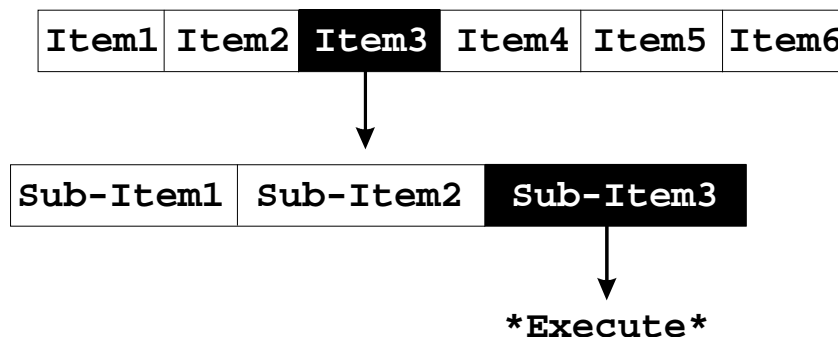


Figure 48: Menu layout better suited for hand gesture.

click and drag can take time to master — the effect is far less dramatic than it is with gesture.

With gestural input the learning curve is steepened by several factors.

- People are not initially comfortable pointing to and gesticulating at a computer screen. They tend to make awkward and uncomfortable poses in the beginning, for example rather than pointing naturally, one user would thrust his shoulders back and point with a straight arm. One result is that the shape of their poses was substantially different than the system had been trained on. User discomfort was aggravated by the monitor near the screen that was used for calibration and debugging and always showed the user the camera view of themselves. It often helped to turn this off once the system was calibrated.
- New users tended to tire more easily holding up their hand. This has several causes. New users tend to be more stiff in their interactions, and the additional muscle tension is tiring. New users take longer to perform actions and often have to perform them multiple times to get the desired result. Finally they simply are not used to holding up their hands in front of them, and it takes time for their muscles to adjust.
- There are no memory aids for what to do next. The user can not simply search for the appropriate menu item, but must remember the sequence of motions and poses. In spite of our efforts to design a natural interaction language, some users still complained that it was difficult to remember.
- The gestures used here are a sequence of motions and poses. Since the current implementation of the system is somewhat rigid in the amount of variation in timing it is willing to accept, the variability with which new users perform gestures seriously degrades recognition performance. For example, when a new user pauses to remember what to do next the system often interprets that pause as part of the gesture and responds with an unexpected action.

In general it was our experience that users could begin to select and move items accurately within a few minutes. Here there are few timing dependencies and the gestures are easy to remember, so the only real obstacle to overcome is the awkwardness of pointing at a screen. The two-step gesture used for operations such as resize and menu selection take more time to master. They are both sensitive to timing and have a longer sequence to remember.

There are several ways to address the learning curve. One is to develop some kind of

user training tools. Training tasks such as moving a ball around the screen, or simple games may help the user adapt to gestural interaction more quickly. This would be similar to the computer games which teach people to type, or the training tapes which came with early Macintosh systems.

An example closer to this work is a new interface device called a Ring Mouse™. This is a ring that fits the user's finger containing an ultrasonic transducer which transmits back to a pair of receivers placed on the top and sides of the screen, allowing the system to triangulate the position of the hand. The ring has two buttons on it which allows it to be used as a direct mouse replacement in PC GUIs. It comes with several demonstrations, including a ring-toss game which allows the user to become comfortable with the device. [Ka95]

It is possible to tune system parameters, such as timing thresholds to make the system better suited to new users' tentative movements. This would require the thresholds to be modified as the user becomes more proficient. It has been suggested that the system may be able to automatically monitor certain characteristics of the user's performance, such as speed of motion, and adjust the parameters automatically. These ideas have not been explored.

Finally, it should be possible to modify the interaction language so that it is more suited to new users. More generally, there could be several different languages that a user could choose between. In the ideal case a user could easily design their own interaction language. While this is theoretically possible now, there are several subtleties in the current language required for reliable gesture recognition. The language would have to be redesigned so that users could easily modify it and expect good results. Again, these possibilities have not been explored.

#### **5.2.4 Design of a Practical Gesture System**

While it is beyond the scope of this thesis to design the next generation user interface, this work has naturally lead to several thoughts on the subject. As a final exercise, this section will speculate on one way a general purpose interface using gesture might look.

The premise of this speculation is that the interface will still be related to current GUIs by evolution rather than revolution. It will remain essentially two dimensional rather than immersive, and it will use metaphors similar to today's interfaces. The target audience will be the same people as today use high-end workstations, and the target tasks will be the range of tasks they perform, word processing, 2D graphics, CAD,

communication, database, spreadsheet, etc.

Hardware technology needed for an advanced interface is becoming common. Multimedia and video-conferencing applications are driving the inclusion of cameras and video-grabbing hardware with workstations. CPUs are getting fast enough to support truly advanced interface styles. Large, flat, high quality LCD screens are becoming available and prices are coming down.

The various software tools needed to support an advanced interface are also starting to appear commercially. Speech recognition, gesture recognition, eye tracking, real-time 3D graphics, OCR and other tools all work in isolated products or prototypes, but have not been combined into a unified whole. Progress is being made on understanding how to integrate these various modalities. Cohen [CS89][Co92] explores the role of natural language in a GUI and concludes that they complement each other well. Hauptmann [Ha89] has shown that people strongly prefer to use speech and gesture together for some graphical manipulation tasks, while Bolt and Herranz [BH92] have built a system to explore the interaction of speech and gesture.

The design for an advanced interface must start at the physical level. The design of a screen and keyboard is outdated and limiting. Consider draftsmen. They sit for hours doing detailed drawings, a task in many ways the physical analogy of today's information worker. The drafting table has evolved as a very comfortable environment for that type of work — large, tilted at an angle to support your arms. Consider a drafting table covered with LCD screen. Even better, make it wrap around like a corner desk. You now have large amounts of screen space, all at a good distance and viewing angle, and the layout naturally supports the body in a comfortable position. Paper documents could be placed on the desk, held either by clips or stops, to be manipulated side-by-side with electronic documents.

How would you interact with such a screen? A keyboard and mouse would not integrate well. Where would you put the mouse? To get from one side of the screen to the other would you have to roll it all the way over? What about its cord? If it has one, it will be getting in the way as you move the mouse around the desk. If it does not you have the problem of the TV remote control — it's never there when you need it. Where would you put a keyboard such that it would not cover the screen but could easily be moved to where it was needed?

Consider, instead, a “deviceless” interface that relies on touch, gesture and voice to completely do away with the need for physical interface devices. Given a drafting table

design, the hands can rest on the screen itself, providing some stability, but rise above it for a wider range of motions. Cameras placed above the work surface and looking down could easily separate the hand from the known background. One problem with this arrangement is knowing when the user touches the screen, so add the ability sense when and roughly where the screen has been touched. Gesture is a good fit with touch screen technology. While touch tells when an event occurred, gesture can tell you what that event was. That can not only help put it in context, but help ignore extraneous touch events, such as the user's elbows.

Voice recognition can take the place of most keyboard tasks, and perhaps additional tasks such as menu item selection and commands when you don't want to take your hands away from what they are doing.

This work has demonstrated that gesture is good at many of the tasks traditionally reserved for a mouse. In the interface, then, gesture would primarily be delegated the tasks of positioning and general object manipulation. For most interactions, simple pointing and gesturing, as has been explored here, would suffice. Rather than mouse gestures like double click or click and drag, hand gestures would be used like grabbing an object to move it, stretching it with two hands to change its size, circling to scroll or flicking your fingers to throw something away. More abstract or uncommon actions would be performed using the menus described above. When greater precision is needed than can be provided by finger pointing, the user could pick up a plain pencil to indicate points on the screen with increased precision.

Others have begun to explore similar ideas. Mapes and Moshell [MM95] used a drafting table as the base for their two-handed object manipulation work. The idea of an electronic desk has been explored by Wellner in [We94]. He put considerable thought into the interface possibilities it presented, but only built a limited prototype, and did not address many of the practical problems involved. Several commercial products have been introduced which present the user with a large projected image covering a desk. The projected image is relatively low resolution compared to today's screens, and these companies generally do not address the interface issues.

