Design Issues for Vision-based Computer Interaction Systems

Rick Kjeldsen Jacob Hartman
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
(914) 784-7558
fcmk, jhartman@us.ibm.com

ABSTRACT

Computer Vision and other direct sensing technologies have progressed to the point where we can detect many aspects of a user's activity reliably and in real time. Simply recognizing the activity is not enough, however. If perceptual interaction is going to become a part of the user interface, we must turn our attention to the tasks we wish to perform and methods to effectively perform them.

This paper attempts to further our understanding of vision-based interaction by looking at the steps involved in building practical systems, giving examples from several existing systems. We classify the types of tasks well suited to this type of interaction as pointing, control or selection, and discuss interaction techniques for each class. We address the factors affecting the selection of the control action, and various types of control signals that can be extracted from visual input. We present our design for widgets to perform different types of tasks, and techniques, similar to those used with established user interface devices, to give the user the type of control they need to perform the task well. We look at ways to combine individual widgets into Visual Interfaces that allow the user to perform these tasks both concurrently and sequentially.

Keywords

Perceptual User Interfaces, Vision-based User Interfaces, User Interface Design, Head and Hand Gesture Recognition.

1. INTRODUCTION

In recent years there has been increasing interest in user interfaces which take advantage of cameras and computer vision technology [1][7][11]. Unfortunately, most work has focused on either recognition of human action, deferring any consideration of how the model is to be used, or of developing interaction techniques in isolation, with little consideration of how the technique may be generalized to other tasks. Much more needs be considered to develop useful interactive systems.

In this paper we will attempt to step back and consider the problem of Vision-Based Computer Interaction (VB-CI) from a wider perspective. The ideas presented here are based on our experience in building several different interactive systems including ones for facial pointing [4] and other head gestures [5], various tools for disabled users such as the TouchFree Switch [13], hand gesture recognition systems [3], finger pointing and selection with projected displays [6] and numerous less successful prototypes. The applied nature of our work has forced us to address the entire range of issues involved in creating practical tools.

By VB-CI we are referring to technology which uses a camera to sense the user's intentional actions and then responds in some way. In a fully functional user interface VB-CI would be only one component of a larger multi-modal user interface, perhaps playing a role analogous to the pointer in the UIs of today by providing spatial and manipulation information.

We envision a VB-CI Interface, or VI, as being composed of individual widgets, similar to how a current GUI is composed of scroll bars, buttons, menus and the like. Each widget provides a basic type of interaction, such as triggering an event or controlling the value of a parameter. A widget may or may not have a spatial location or a representation on the screen. Just as with current interfaces, when the task changes the set of active widgets changes as well.

This paper will address various aspects of the design of a VI. It will first consider general issues such as the interaction styles suited for direct sensing interaction. Vision-based user interactions have different strengths and weaknesses than traditional input devices. These differences must be taken into account when considering how they should be used and what the resulting interface will look like.

Next, we will examine the problem of designing and building individual interface widgets; the atomic actions like pointing and scrolling that make up a user interface. We will consider some of the constraints on the control movement imposed by human factors, the types of control signal that can be derived from those movements, and how those signals can be used to perform the desired task.

Finally we will address how these widgets can be combined into a complete VI, and how a VI can be integrated with traditional applications and control mechanisms.

We hope that this discussion will provide some insight into what tasks are well suited for direct sensing interaction, the range of ways tasks these tasks can be performed, and how these tools can be provided to the user in a practical package.

The discussion will be in the context of vision-based user interface techniques, but much of what is said applies to interactions using other "direct sensing" techniques, such as sensor equipped clothing, active sensors like sonar and IR, etc.

It is important to keep in mind that our focus is on Interaction rather than Recognition. In other words, we are not interested in modeling human behavior or even recognizing it beyond what is needed to control some aspect of a computers activity. In some cases, an explicit model of the user or their activity may be the best approach, but in many cases a more direct mapping from some image feature to the control task is appropriate.

2. INTERFACE DESIGN

2.1 Interface Style

A traditional pointer can sense action along two dimensions, and even this limited expressiveness is not fully utilized. In the nearly ubiquitous point and click interface, the sensed activity is only used to identify a location on another 2D plane, the screen. Using that location and a combination of clicks, all types of tasks are performed.

With VB-CI we have the potential to extract a control signal with many more degrees of freedom, so it does not make sense to limit the interpretation to a screen location. Of course, it is possible to translate the control signal into a location and for some tasks that may be the best way to use it. For tasks whose primary goal is selection or parameter control, however, it often makes more sense to apply the sensor data directly to the task. For example, scrolling speed could depend on the angle of the user's face as it aims above or below the display window. This interaction saves the screen clutter required by scroll bars, and saves the user the distraction of having to place a pointer in a small rectangle to scroll.

When taking advantage of the expressiveness of gesture-based interaction, it is important not to go too far. As an extreme example consider an interaction based on a complex sign language. This would be more expressive than most tasks require. There would be a large penalty in terms of the effort required on the part of the user for learning. There would likely be problems with remembering the signs to perform for rarely used tasks. The recognition of such a language would require significant computing resources.

A great deal of knowledge of human computer interaction is embodied in the form of current graphical user interfaces. We therefore strive for a middle ground, where tasks which have proven valuable in current interfaces are performed using interaction styles based on a more flexible direct sensing techniques.

2.2 Interaction Tasks

In order to explore the ways in which gesture-based interaction can be used in a user interface, we have clustered tasks by their purpose. In previous work [5], we identified four categories of task well suited for head gesture:

- **Pointing:** accurately identifying an arbitrary location on the display with high resolution. This can be used to control a traditional cursor, or to position some other object on the screen. In addition to the familiar direct manipulation tasks, pointing can also be used as an intermediate step for higher-level interpretation.
- **Parameter Control**: determining the value of some continuous parameter. A common example is to control the location in a file to be displayed (scrolling), but numerous other applications exist such as setting the pitch or duration of a note.
- **Spatial Selection**: identifying one of several spatially distributed alternatives. For example, you may tilt your head left to select the left button in a dialog box or reach out your hand to touch a target floating off your right shoulder. The alternatives may be distributed in image, user or screen space.
- **Symbolic Selection**: identifying one of several alternatives by non-spatial means. The most obvious example is to shake your head for yes or no. The result could be a simple binary signal, or could use a more complex alphabet.

When we consider hand, as well as head gestures, the fourth category, Symbolic Selection, can be expanded to include a wide range of symbolic activity. This topic is beyond the scope of the current discussion, but the first three categories will form a good basis for our examination of VB-CI in general.

2.3 Context

With any complex interface there are likely to be far more tasks that may be performed than individual control actions. This creates an ambiguity as to the user's intention, which must be resolved by contextual information.

A strong sense of context can also help avoid what is often called the Midas Touch Problem, where everything the user does is interpreted as an interaction. In traditional user interfaces, the user can simply not touch an interface device when they don't want to use it. With direct sensing, however, the user's actions could potentially always be "active". Context can tell us when to attend to the actions and when to ignore them.

In current GUIs, context is provided by the screen location of the pointer. For example, a mouse down event in a scroll bar guides the interpretation of a subsequent drag as being intended to control the location within a file. A mouse down event in title bar implies a subsequent mouse drag should control the location of a window. A click in an inappropriate location can be ignored.

In VB-CI, it is also possible to use the spatial location of the interaction to provide context. We refer to these interactions as "target-based", implying there is some spatially located target where the action must be performed. As an example, a user may have to place their hand into a rectangular region beside them and then move it up and down in order to control a parameter value, creating a widget similar to a scroll bar.

VB-CI widgets, however, can often be spatially independent. For example, a selection task such as selecting the leftmost dialog button may be implemented by having the user tip their head in that direction. In this case, the location of the head does not matter, only its motion, and the context must be provided by some other means, generally by the state of the application. .

One can also envision an interaction where the context is provided by the current state of the user. For example, the shape of a hand during a motion may control what task the motion performs.

2.4 Image Space and Screen Space.

One concept that must be considered when thinking about a VB-CI is the reference system that is being used for spatial reasoning at any point in time. When we say a target is somewhere in space we have to be clear about what space that is. If it is screen space, we have to map the user's actions to the screen before interpretation. We can stay in image space, so that the target is always at the e.g. left of the camera's field of view, or translate to user space, so that the target is always some distance to the left of the user.

Traditional pointing devices always map to a location in screen space. Gesture can also be mapped to screen space, giving a relatively traditional point and click style interface, but that is not always needed. Direct interpretation of the action in user or image space often provides a simpler solution in terms of both the computation involved, and the user's perception as well.

Keep in mind that the transformations between these spaces may be very simple. In the case when a camera is observing a user's hand in front of a projected display, image space and screen space can be essentially the same. If the camera is observing the user and they are relatively still, image space and user space are very similar. This implies that in some circumstances we can reason directly in image space and so avoid the complexity of transforming from one space to another.

3. DESIGNING WIDGETS

We will now turn our attention to how individual VB-CI widgets should be designed. Three aspects must be considered: the **Control Action** used, that is the motion the user makes to affect control; the C**ontrol Signal** to be used, meaning the signal extracted from the control movement that will be actually used to control the task; and the **Transfer Function**. This is the algorithm by which the control signal is converted to the task.

3.1 Control Action

The choice of a control movement for a task is often more art than science. It is influenced, however, by several constraints including:

- **Intuition**: How natural is the motion for the task? Is it easy to remember?
- **Motion**: Does the user have sufficient range of motion to provide the need resolution?
- **Stability**: Can the user perform the movement with sufficient stability to accurately control the task?
- **Comfort**: Can the user comfortably perform the task for the length of time, or the number of repetitions required?
- **Multiplex**: Can the user perform the motion while doing everything else they need do at the same time? For example, can they comfortably see the feedback the system provides?
- **Visibility**: Can the system see the movement adequately?
- **Midas**: The action should be unique enough that it will not be accidentally performed by the user, thus avoiding the "Midas Touch" problem.

With these constraints in mind, consider what might be good control actions for each of the task types in Section 2.1:

3.1.1 Pointing Actions

The most intuitive pointing actions include aiming some body part, such as a nose or a finger, at the desired location. While it is possible to map any 2D motion, such as a hand on a desk, to a screen location, when you do often much of the advantage of gestural pointing over a device such as mouse disappears. A stable and responsive pointer motion is essential. When a pointing task requires the user to repeat the action often or for long periods, the motion must be comfortable and generally the hand should have some type of support.

If the head is used, it should always be balanced comfortably over the neck and not tipped at an awkward angle. In an early head pointing system, we tried left/right head tilt because the motion is easy to recognize. Unfortunately, the repeated and extended tilting required quickly became uncomfortable. The lesson: excursions in tilt should be limited in amplitude, duration and frequency because in tilt the head is not balanced over the spine, making the neck muscles work to hold it up.

A natural choice for head-based pointing is facial aiming. Accurate aiming is intuitive as people aim their faces for social reasons as well as to support acute vision and hearing. As the feedback is strictly visual, the multiplex constraint imposed by the need to watch the pointer is important to address.

3.1.2 Parameter Control Actions

Parameter Control tasks are generally one-dimensional and the interactions short, which can make finding an appropriate control motion easier than with pointing. The intuition of the motion should correspond to the specific task, such as up for increasing values, down for decreasing. Stability can be an issue to achieve good accuracy.

For a task like facial scrolling, a natural choice is again based on facial aiming, where the user aims their face above, below or to the side of the document to be scrolled.

3.1.3 Selection Actions

Spatial Selection tasks are generally very brief, making stability and comfort even less of an issue. This brevity can make multiplex constraints an issue, however, as the user may not want to move their hands from their current task (e.g. typing) for such a brief interaction. For this reason, facial gesture can be a good alternative for many selection tasks, the user aims their face in the direction of the selection. The range of motion need only be large enough for the system to unambiguously distinguish the alternatives. Depending on the number of alternatives the motion can be very small.

As an example, consider using facial selection to select one of buttons in a dialog box. A natural interaction is to have the user aim their face left or right to highlight the desired button, then tip their head down to select it. The down movement is important to avoid an inadvertent selection when the user paused to think about their response.

3.2 Control Signal

Once a control action is determined, a signal must be extracted from the image sequence that captures the important aspects it. This signal is then used to drive the desired system response. This

paper will not attempt to classify all the different types of control signal that could be used. The state of the art in gesture recognition is changing very fast and much of what we say would quickly be outdated. Instead, we will try to classify control signals along general dimensions.

3.2.1 Image- versus model-based control signals

Much work in the recognition of human action comes from the computer vision community where building accurate internal models of the world is a strong tradition. Once a model is built, some parameter of that model can be used as the control signal.

The model-based approach often extracts more information about the user than necessary, implying it is more computationally expensive than necessary. In addition, current techniques for building 3D models are often noisy and unreliable with respect to the demands of real time interaction [12].

Some of these problems will fade with time, but the advantage of decreased computational complexity should not be underestimated. Higher frame rates contribute directly to better usability; moreover, camera-based interactions are likely to be used in concert with other resource intensive applications such as voice recognition as part of a multi-modal user interface. Even with the faster machines several years in the future, users will want their computing resources available for the target task, not squandered by the interface.

For real-time interactive systems, the simpler image-based approach is often preferable. Here the image sequence is examined for a signal that tracks the control movement, and then mapped directly to the task.

Image-based control signals fall into two categories, position-based and value-based. Value-based signals track the value of some image parameter at a location. The signal is chosen such that its value changes distinctively when the user performs the control action. For example, you might compute a color-histogram of a small region of the image to detect when the user "touches" it. This approach can work well for detecting discrete events, but since these signals generally do not change monotonically with the user's actions, it is difficult to use them for parameter control type tasks.

More interesting are the position-based signals, where some image feature is tracked within the image. We refer to these as 2D control signals. These signals generally do vary monotonically with the user's actions, making them suited for control tasks. They can also be interpreted with some decision function to give a selection task.

3.2.2 Absolute Versus Relative Control Signals

Many position-based control signals can be considered as belonging to one of two categories:

- **Absolute signals**, which record the state of the user with respect to some fixed reference.
- **Relative signals**, which record the state of the user with respect to their previous state.

These can correspond to location and velocity, or to orientation angle and angular velocity, or to nearly any other image feature we may wish to extract.

Relative signals are difficult to use in situations where there is a limited range of motion or where there must be some relationship between a user's position and the state of the system she is trying to control. For example, with facial aiming the need for the user to look at the current pointer location as they control it makes relative signals impractical. Absolute signals have their own set of complications. A reference state must be determined, then either remain fixed, or be tracked during the interaction. For example, depending on the task we may want to use the location of the hand within the image, with respect to some background object, or with respect to the user's body. In the later two cases the location of the reference point must be tracked, in addition to the location of the hand.

An example from our facial pointing system will illustrate the discussion. Because of the importance of real time response, we chose to use an image-based control signal. If the camera is looking at the user approximately head-on, small 3D head rotation appears as 2D translation of the facial image (with increasing amounts of image warping). The expected range of facial image motion can be computed to within small error using the size of the facial image and typical head dimensions. This allows the location of the facial image within its range of motion to be used as a proxy for head rotation.

Using image-based control signals allows us to ignore several difficult problems we would have to address in order to build a model. For example, we estimate the range of motion of the face based only on it's size. We need not care if the apparent size is due to the distance of the user from the camera, the properties of the lens, or the resolution of the image. What is important is the relative position of the user's face within its range of motion, giving us an absolute control signal.

We track the face using cross-correlation: matching a face template within a small search region in each frame. The average of the absolute value of the difference between the gray level of corresponding pixels seems to give better stability than using a squared pixel difference.

The search region is computed using the maximum reasonable head speed during pointing, determined empirically to be about 3 facial diameters per second, and the instantaneous frame rate. The search region is expanded on two sides, if needed, to ensure that it includes the center of the range of motion. This gives the user a simple recovery mechanism when the system looses their face - orienting their face back to the position used in training (looking at the center of the screen) usually results in the face being found immediately.

This relatively simple template-matching tracker has proven robust under a wide range of circumstances. In good lighting, we can track the face through about a 60-degree arc horizontally (30 degrees on either side of the camera), and through about 40-degree arc vertically. Tracking speed is excellent. At 160x120 pixel resolution we can track the user's face at nearly 30 frames per second, using only a small fraction of the CPU[1]. With a well-designed transfer function, this resolution is ample for smooth and accurate pointer movement.

[1] All performance numbers were obtained on a 700 MHz Intel processor running Windows 2000. Code was written entirely in C++ using only basic optimization techniques.

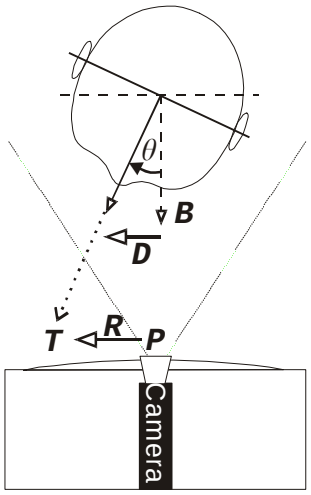

Figure 1: Configuration of user and screen for facial pointing.

3.3 Transfer Function

The final step in building a widget is to map the control signal to the task. For pointing and parameter control tasks the transfer function is generally a mapping from one continuous value to another. For selection tasks, it becomes a decision function on the control signal.

3.3.1 Transfer Functions for Parameter Control Tasks

A transfer function for parameter control tasks needs to do at least three things, reduce noise and error in the sensed signal, compensate for low control signal resolution, and provide the user with a pleasing and usable response. The jobs can be addressed independently, or combined into a single function, but all aspects must be considered. Noise reduction is often addressed in current "gesture recognition" work by filtering approaches [11]. Low sensing resolution is generally handled in the same step. Issues of usability are only rarely addressed in the PUI community. Fortunately, we can draw inspiration from the traditional HCI community, where these issues have been addressed for some time [9].

3.3.2 Rate vs. Position Control

The value of a parameter, such as the location of the pointer on the screen can be controlled using either its change (motion) relative to a previous value (rate control) or by setting its absolute value with respect to some reference (position control). Consider two examples. They use two different absolute control signals in order to make them more realistic.

Absolute Signals controlling Position (AP): The location of the facial image within a box positions the pointer on the screen correspondingly.

Absolute Signals controlling Rate (AR): The rotation of the face from straight ahead determines the speed of the pointer in the corresponding direction.

True nose pointing, in which the pointer is positioned where a really long nose would touch the screen, is an instance of AP where the mapping function from orientation to screen location takes into account the 3D geometry of the environment.

Depending on the task and the desired interaction style, either rate or position control may be applicable. The following will discuss our development of the transfer function for a face-tracking pointer in order to make some of the issues more concrete.

For what follows, *C* refers to the control signal, which can be either the angle of the face, q , or the displacement *D* of the facial image within its range of motion as described in the previous section. *P* refers to the location of a pointer on the screen (see Figure 1). Every frame the transfer function will convert *C* to a new *P*.

In our first attempts to build a transfer function for a facial pointer, we mapped *C* by various simple transforms to either an absolute pointer position *T* (AP interaction) or a pointer motion R (AR interaction). The AP version required significant filtering of *T* to compensate for noise in and coarse resolution of *C*. While this approach worked, it did not give a pleasing and responsive pointer motion, which had a big impact on usability and user satisfaction.

With the AR version, we were able to tune the transfer function between *C* and *R* to improve pointer dynamics. This gave a transfer function which was suitable for various other body parts, e.g. hand tracking, but with face tracking we encountered a problem. Under common circumstances, the face must be aimed away from the pointer to get the desired motion, making viewing the pointer uncomfortable. For example with *P* on the far left of the screen, the user has to aim their face straight ahead to keep it still and look at *P* out of the corner of their eyes. To move *P* slightly right they must aim their face further away from *P*, making matters worse. We were able to compensate by applying a bias to the facial offset calculation, such that *C*=0 when the user's face aimed at *P*. Unfortunately, this made the algorithm cumbersome to work with, limiting further development.

This, however, lead us to a hybrid rate and position control scheme, where *C* is translated to a target screen location *T*, and the distance *F* between *P* and *T* determines *R*, which is assumed to be in the direction of *T*. The nature of the mapping from $F = |T - P|$ to *R*, more than anything else, impacts usability, and here much can be learned from similar work in the HCI community such as [9]. Our best mapping to date is from a sigmoidal relationship

$$R = \frac{R_{\max}}{1 + e^{-\left|\frac{F-k}{\mu}\right|}}$$

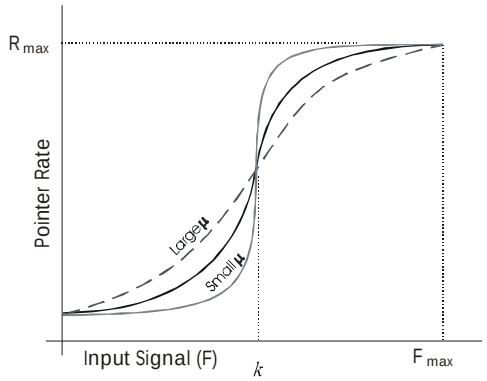

Figure 2: Sigmoid Transfer Function

Figure 3: a) Camera view of user interacting with projected button; b) image difference data; c) overlay of search region (square), button active area (circle), and the fingertip template shown at the pointing location.

such that when T is near P large head movements are needed to move it, making fine positioning easier. When $|T-P|$ is large, $R \simeq T-P$ so that P tracks T very closely, making for rapid long distance moves. Importantly, a sigmoid is easy to tune by adjusting knee κ, slope μ and the maximum output value $R_{\max}$.

The main advantage of this hybrid algorithm is that it allows us to separate issues specific to sensing the control signal from issues of pointer dynamics and usability. This has made it easier for us to both tune the system for good usability by adjusting pointer dynamics, and also to make progress on the remaining problems such as compensating dynamically for changes in user / screen geometry.

Pure AR interaction does have its place. It is well suited for input signals that do not map well to a screen location, making it suitable for use with a wider range of human motion. It is also easier to adapt for use with relative control signals. For facial pointing, however, the hybrid method is easier to work with.

With respect to the preceding discussion, then, we can examine how this algorithm addresses the various tasks of a transfer function. The resolution mismatch is handled by the fact that we are controlling the rate of the pointer, not its location. Low amplitude noise is addressed by the low "gain" between small control signals and pointer motion (the left of figure 2). High amplitude noise still gets through. This generally does not present a problem, but if it did in a particular application, it could likely be handled by specific filters in the computation of *T*.

This algorithm addresses several usability issues. It addresses the multiplex problem of having to look at the cursor with the position control component (the fixed computation of *T*). It also addresses an inherent conflict between providing the user with precise control for fine positioning and at the same time fast long traversals using a highly non-linear gain between *F* and *R*.

One more aspect of the hybrid transfer function should be pointed out. There is another degree of flexibility by which this algorithm can be adjusted to the needs of the user and the task. If the horizontal and vertical components of *R* are computed independently, the pointer will show a mild affinity to track horizontal or vertical lines. This comes about because the two components of *C* fall at different points of the sigmoid. This affinity can be helpful when the pointer is being used primarily as a selection device, but when the user desires more accurate free form drawing, it can easily be disabled by computing *R* using the combined vector length of *C*.

3.3.3 "Transfer Functions" for Selection Tasks.

Transfer functions for selection tasks are less of the form of a numerical mapping and more of the form of a decision function.

For image-value-based control signals (see Section 3.2.1) the decision function is generally of the form: "has *C* changed sufficiently in this region to justify a decision". This decision can be based on many types of comparison. One which we have had good results with is color histogram matching. Here the color histogram of the desired image region is matched to the histogram of either the background to determine when sufficient change has occurred, or to the foreground, obtained through training, to determine when the desired action has been performed. The distance threshold gives a simple knob for the end user to adjust to obtain the desired sensitivity. The histograms are structured to remove some of the dependence on lighting, generally by using HIS color space, and quantizing the Intensity dimension very coarsely. This is the approach taken in the TouchFree Switch [13].

Position-based spatial selection tasks often require *C* to be first converted to a space more intuitive for the user than the original image coordinate system before being operated on with a decision function. The conversion can be done using simple coordinate system transformations, or by complex transfer functions like those described in Section 3.3. After the transform, however, the choice of decision function depends heavily on task and user constraints.

In a projection display interaction system we have developed [6], the user's fingertip is located in an image of a projected display (Fig. 3a) by matching a fingertip template in a frame-to-frame difference image (Fig 3b & 3c). The trajectory of the fingertip is then examined for patterns that indicate the user reached out, touched the active area (button), paused, and then retracted. This decision function proved to be very robust in the target domain, where users performed isolated selection interactions, but would fail in an environment where sequential selections were needed with no retraction in between. This illustrates how the decision function must be tightly tuned to the user's actions in the given environment.

4. APPLICATION ISSUES

Once we have a library of widgets we must address how they are combined into sets that allow the user to perform all the tasks required by an application at any point in time. We must be able to:

- Create collections of widgets (configurations) to perform application level tasks.
- Switch between configurations to accommodate changes in the application's context.
- Allow the user to customize the interface to their desired motion patterns.
- Adapt, either automatically or with user assistance, to the current imaging conditions.
- Easily integrate VIs into a new or existing application without any knowledge of computer vision.

We have just begun to explore these issues, but already some important points have become apparent

4.1 Configuration Sets

Like a standard GUI, a VI must be able to handle context changes in the application level task. We handle this by activating the appropriate configuration set at the request of the application. For instance, when a yes/no dialog appears, an application may temporarily replace a face-tracking configuration with a symbolic widget that interprets nods or shakes of the head. In this way, visual interface widgets can be combined both concurrently and sequentially to provide more complex interactions.

One problem that must be addressed in any practical system is that the user be kept aware of which widgets or configuration sets are active any time. One solution is for each widget to have an on-screen representation drawn over the video stream in a display window. This approach works well when the camera view is looking straight back at the user such that the display appears like a mirror. In other circumstances, however, the camera is at an odd angle, and the display is disorienting to the user. In this case we have found it very useful to be able to project a target or other widget representation directly into the environment for the user to interact with (see [6]).

Creating configurations of widgets with completely independent control signals, such as combining selection targets with facial pointing, is relatively straightforward. There is little conflict between widgets, and the user can easily understand what they need do to activate one widget rather than another.

When the widgets share a control signal, however, the situation becomes more complicated. A good example comes from a system where we tried to combine facial pointing with an on-screen keyboard where the characters were selected with left / right facial motion, and typed with a vertical movement, up for upper case, down for lower case. Both the system and the user needed to clearly differentiate when the facial motion was to be used for pointing or typing.

Our solution was to give one of the widgets a spatial trigger. The keyboard only became active when the pointer reached the top of the screen. Then the keyboard was displayed and the keyboard widget took over. We are in the process of formalizing a method for this type of control signal sharing between widgets.

In order that widgets in the same configuration be able to share control signals, widgets must not be built as single, monolithic objects. If there are two separate widgets driven by a face-tracking algorithm (such as the pointer driver and on-screen keyboard), there should be no need to run the tracking algorithm twice. We have begun to explore how widgets can be divided vertically to facilitate this type of sharing.

To handle these related design issues, namely the multi-staged processing involved in a widget and the one-to-many relationship that may occur between control signal detection algorithms and widgets, we build widgets from "stackable" layers of components. A component is an object that handles a single processing task and passes the resulting control signals to the component(s) in the next layer. Expanding our previous example, a component would encapsulate the face-tracking algorithm, inputting the raw video stream and outputting the position of the user's face within the image. These face position signals would subsequently become the inputs to two separate chains of components, one of which would make up the pointer driver and the other comprising the on-screen keyboard.

This hierarchical component architecture also has the advantage of making the widgets very versatile. A developer (or even a user) can create a new widget by mixing and matching existing components. For instance, a face tracking widget that drives the cursor can be altered to drive scroll bars simply by swapping in a component that sends scroll messages rather than mouse messages.

4.2 Training and Personalization

Unlike a typical physical control device (i.e., a mouse or keyboard), the vision-based interface must derive its control signals from a noisy, high-bandwidth input medium. Equally important, every person has somewhat different movement patterns and preferences.

Differences between users tend to be filtered out by physical user interface hardware. If someone has difficulty using their right hand, they can move the mouse to the other side of the keyboard without the computer caring. With direct sensing interfaces, however, the differences in how people move must be handled explicitly by the software.

Because much of our work thus far has been focused on the special needs community, where the ranges of motion and levels of control differ significantly among these users, we have been forced to address this issue in its most extreme form.

To address inter-personal variation, personalization and training of a VI become critical. We allow two stages of personalization. At the high end, the user can move widgets and customize their response, even changing the widget type from a selection pallet if needed. This type of configuration is not something that should have to be performed every session, or by every user, but in our experience having this type of flexibility is important I achieving the potential of the system.

After a user personalizes her interface configuration, a training step is often required, where the user shows the widget what type of action they will perform so that it can tune its recognition procedure. Completely automatic training is desirable but is often difficult and unreliable in the current state of the art. In our experience, manual training can be made sufficiently painless -- often reduced to simply pressing one or two keys (or the

equivalent) at the beginning of a session – that fully automated training is not necessary for usable systems.

5. CONCLUSION

This paper has endeavored to outline the space of vision-based computer interaction, identifying various dimensions along which vision-based interactions can be built. We have classified the types of tasks which seem to be well suited to gesture-based interaction into the categories of Pointing, Parameter Control and Selection, leaving aside Symbolic interpretations for now. We described the design of widgets to perform each type of task, starting with the choice of control movement, then the extraction of control signals from the visual input, classifying them as Absolute or Relative. We described transfer functions for the various types of tasks, including Position and Rate control functions for parameter control tasks, and decision functions for selection tasks. Finally, we addressed how to combine widgets with each other into sets that can be enabled and disabled to suit the needs of the application. Along the way, we discussed the ways in which gestural interaction differs from traditional user interfaces, and gave examples from real systems.

We hope this will provide the beginnings of a framework for a more rigorous analysis of gestural interaction systems. We can now begin to see the parts of the solution space that have and have not been explored, and to identify promising untried interaction types. We will also be using this analysis to design more formal user studies that can be used to quantify the differences between the various approaches.

6. REFERENCES

[1] Communications of the ACM, Vol 43. No. 3, March 2000, M. Turk ed.

[2] Crowley, J., Coutaz, J., and Berard, F., "Things that See: Machine Perception for Human Computer Interaction", Communications of the A.C.M., Vol 43, No. 3, pp 54-64, March 2000

[3] Kjeldsen, F., *"Visual Recognition of Hand Gesture as a Practical Interface Modality,"* PhD Dissertation, Columbia University, 1997

[4] Kjeldsen, R., "Facial Pointing", presented at the 4th International Workshop on Gesture and Sign Language based Human-Computer Interaction (Gesture Workshop 2001), proceedings forthcoming

[5] Kjeldsen, R., "Head Gestures for Computer Control", in the Proceedings of the Workshop on Recognition And Tracking of Face and Gesture – Real Time Systems (RATFG-RTS), Vancouver, BC, Canada, July 2001

[6] Pinhanez, C., et.al., "Transforming Surfaces Into Touch-Screens", submitted to CHI 2001 (system demonstrated in the Emerging Technology section of SIGGRAPH '01, Los Angeles, CA)

[7] Proceedings of the Fourth International Conference on Automatic Face and Gesture Recognition, 28-30 March, 2000, Grenoble, France. IEEE Computer Society Order Number PR00580

[8] Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI '98), San Francisco, CA, Nov. 1998, M. Turk editor

[9] Rutledge, J. and Selker, T., Force-to-Motion Functions for Pointing, in the Proceedings of the IFIP TC 13 Third International Conference on Human-Computer Interaction, August 1990 (Interact '90)

[10] Toyama, K., "Look, Ma - No Hands! Hands-Free Cursor Control with Real-Time 3D Face Tracking" in [8].

[11] Wu, Y. and Huang, T., "Vision-based gesture recognition: A review" in Lecture Notes in Artificial Intelligence V1739 1999

[12] Wu, Y. and Toyama, K., "Wide-Range, Person- and Illumination-Insensitive Head Orientation Estimation", in [7].

[13] TouchFree Switch, a camera-based switch for physically disabled users, released by Edmark. See http://www.edmark.com/prod/tfs/