RC23560 (W0503-056) March 9, 2005
Computer Science

IBM Research Report

Exploiting the Flexibility of Vision-Based User Interactions

Rick Kjeldsen
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Exploiting the Flexibility of Vision-based User Interactions

Rick Kjeldsen

IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
fcmk@us.ibm.com

Abstract

The expressiveness and subtlety of hand gestures, combined with the freedom of visual gesture recognition provides the potential to create user interfaces of unprecedented bandwidth and flexibility. Without some type of structure, however, these very aspects can make gesture-based interfaces impractically complex for both the user and the interpretation system. We will elaborate on this idea, then present a vision system that provides a well defined framework for creating gesture-based interfaces for realistic applications, while retaining much of the flexibility that makes such interfaces so attractive. This framework, using configurations of targeted gesture widgets, shares many similarities to existing user interface design concepts, thus easing the transition to gesture-based interfaces. We finish by describing an extension to that framework which places the design and layout of the interface in the hands of the user, rather than the application designer with a novel approach based on visible artifacts that can be placed in the environment to define the gestures which make up the interface.

1 Introduction

Vision-based user interfaces (VB-UI) have been a hot topic for several years with a devoted following (Davis, 2002), (Turk, 2001), (Wu & Huang, 1999). The idea that nearly any user action can be detected with a camera, interpreted and used to control an application has strong appeal. It has the potential to make human-computer interactions more natural, less rigid, less location specific and more available to users who, temporarily or permanently, may have difficulty using traditional input hardware.

A great deal of the work with VB-UIs has focused on the ability of vision systems to interpret either natural human activity or complex gesture-based languages in an attempt to leverage the free-form nature of vision-based interaction. Our experience, however, suggests that when a gesture-based language is too free-form it presents a host of problems for both the user and the interpretation system. We have therefore taken an approach where the interaction is assembled from fundamental components, similar in many ways to the components underlying current mouse-based user interfaces. This provides both man and machine a solid foundation of known primitives from which to build an interface, but still allows that interface to adapt to the current needs of the user and the environment.

Hand gestures are generally thought of as sequences of hand motions and poses, and most vision-based gesture recognition is based on techniques for recognizing, then interpreting the shape and/or motion of a hand, often in three dimensions. This approach provides great flexibility, an arbitrary gesture at an any location can have meaning, however it requires significant overhead on the part of both the user and the system. The user must learn and remember sequences of motions and poses, then perform them smoothly. For a complex interaction this may be a daunting task. Consider the effort it takes to learn human-to-human sign languages such as Americal Sign Language. The vision system must be able to locate and track the user's hands at an any location, and determine their pose. Then it must interpret the sequence of motions and poses in the context of the current task. Supporting this degree of flexibility can lead to issues like the "Midas Touch" problem, where incidental movements may be interpreted as commands, and to difficulties segmenting one gesture phrase from another.

An alternative approach is to create an interface based on a user's interaction with objects in their environment. Touching the phone might dial the selected number in a phone book application, a piece of paper might serve as an impromptu touch pad, or a control panel of interactive buttons and sliders could be projected on the coffee table.

Interacting with visible objects grounds the interface for the user, making it easier to remember how to interact with the system. Such "targeted" interactions also provide constraints that make it easier for the system to interpret the user's actions. Hand movements are only salient when they are performed near one of these objects, and the types of interactions that can be expected can be limited by the type of object being interacted with.

A complex user interface can then be created by monitoring simple interactions with a complex set of objects. These objects can be static, say a pattern printed on a kiosk, or dynamic, projected into the environment by the system. In the later case we have a situation very similar to current computer interfaces, where the system presents the user with an assembly of interaction components suited for the current context. Each determines one or more parameters based on simple interactions with the user, and provides those parameters to the application. For example, sliding a finger along a line can provide control of a 1D parameter, which could control an action such as scrolling in an application.

The next section will describe systems which use this modular, targeted approach for building VB-UIs, give some details on how they operate and demonstrate some of the applications they can support.

These interfaces, however, are still under control of the application. The application designer must determine in advance which gestures are to be used for what purpose, and lay them out in what they hope is a logical and usable configuration. When the system is set up, this configuration can be adapted to the environmental conditions, but once it is running, the user interface is generally fixed.

Our most recent work has explored how we can place much of the control for the layout of the interface into the user's hands. Starting with modular target-based interactions, then adding the capability to identify visible artifacts in the environment that serve as targets, and a method to associate interactions around those locations with specific functions, we can provide the user with a natural way to create an interface for an application. By laying out objects in the environment the user can define a set of gestures which meets their current requirements and limitations, then change that interface as needs arise. Section 3 will elaborate on this idea.

2 Modular, targeted, vision-based interfaces

There are many current systems which allow creation of vision-based user interfaces. In most cases the system recognizes some hard coded set of motion/pose gestures. We are aware of two recent systems which use the modular targeted approach advocated above. As with traditional computer interfaces, these systems allow the application designer to assemble an interface from a set of predefined components, but in this case each component is a gestural interaction with some target.

2.1 VB-UI systems

In (Ye, Corso, Burschka & Hager, 2004), the authors describe system built to recognize targeted gestures, such as a touch, using an approach they call Visual Interface Cues (VICs). VICs are easy-to-extract image changes which provide cues to the presence of a user's hand under various circumstances. By monitoring sets of these cues in small local image regions, they gather evidence to the presence and behaviour of a user's hand at a specific location. Using an HMM, they detect sets of VIC responses across a grid of regions that suggest a specific gesture about a location. With this approach they suggest it is possible to reliably detect complex local user actions which could then be combined into a more complete interface.

EDVision (Kjeldsen, Levas & Pinhanez, 2004) is a system created at IBM Research to support projected user interfaces. An application creates an interface from *configurations* of interactive *widgets*. Each widget, such as the one described in section 2.2, detects a specific gesture at a target location. A configuration can be moved between predefined surfaces in the environment, and individual widgets aligned with visible targets. The targets are often icons projected on surfaces by a complimentary projection system, but can be any object or location known to the user.

To define and control an interface, an application communicates with EDVision using the XML API described briefly in Section 2.4. Based on a configuration description, EDVision assembles a set of image processing

components that implement that collection of widgets. To change the interaction, an application can send a new configuration description at any time. When a widget recognizes a user interaction, EDVision can either generate an XML message back to the application, or simulate a mouse or keyboard event directly to the operating system.

EDVision was designed to make it relatively simple for a non-vision expert to use. The individual interactive components, the method of assembling an interface from them, and using events to signal user actions are all intentionally similar to existing user interface programming methods. As much as possible, environmental variation is hidden from the application programmer. Some environmental variation, such as lighting, is handled by careful design of the image processing algorithms. The widget described in section 2.2 has been designed to be robust in the face of variations in both lighting and viewpoint. Because interaction takes place with respect to targets in the environment, another source of variation is the location of the widgets in the camera's field of view, which depends on the relative geometry of the camera, the user and the targets. EDVision provides for the deployment of an interface onto arbitrary planar surfaces viewed from arbitrary camera angles. The parameters of the surfaces where the interface can be realized are calibrated when a system is installed at a location, and stored independently of the definition of the interface itself. Section 2.3 will describe this process in more detail.

2.2 Targeted Interaction Widgets

We have implemented numerous widgets that work with EDVision. We classify them according to the dimensionality of the control signal they can generate. Zero dimensional widgets generate a one-time event when triggered by a user action such as a touch. One, two and three dimensional widgets generate control signals with the corresponding number of dimensions in response to user inputs such as the location of a fingertip along a line, within a region, or with respect to a point. The control signals from these widgets can either be returned to the application or used to generate events in the native operating system. Previous publications describe several different types of widgets (Kjeldsen and Hartman 2001, and Kjeldsen, Levas & Pinhanez, 2004). This section uses our latest touch detection widget, not previously described, as an example of a target-based interaction.

Most of our previous widgets were based on tracking parts of the user's body, such as a fingertip, and interpreting that motion with respect to the widget location. Over time this approach proved to be less than ideal. It is easily fooled by common events, such as objects passing in front of the target, it is computationally expensive, and to work best it requires prior knowledge of the user's expected location and size within the image. In recent work we have developed a set of widgets based on a different technique, which we refer to as Polar Motion Maps.

A Polar Motion Map (PMM) is a polar representation of the motion energy around a point in the image that makes important aspects of the motion easy to interpret. The term Motion Energy refers to the pixels in an image which have changed, presumably due to movement in the real world. Specifically we label each pixel where $|P_t - P_{t-1}| > T$ where P_t is the pixel value at frame t, $T = V_{max} * P_{me}$ is a threshold, and V_{max} is the maximum pixel value. The value of P_{me} can be varied to compensate for environmental conditions such as lighting conditions. The resulting image of the thresholded pixel-wise differences will be referred to as a Motion Mask. We reduce noise in the mask using a series of erosion and dilation steps and a vibration desensitization algorithm that removes constantly changing pixels. Clearly there are other methods of finding the motion energy in a scene that could be used as input for a PMM, including approaches based on optical flow and background subtraction. We have chosen this approach for its simplicity and suitability for our application domains.

In the motion mask, moving objects create clouds of motion energy, usually clustered around their leading and trailing edges, in an other wise blank background. Figure 1 shows a motion mask of a user touching a paint can with their outstretched finger. The faint outline of the can is seen as it moves under the touch, as well as points of motion energy from the user's head.

Figure 1: Motion Mask of user touching bucket.

A Polar Motion Map is a polar representation of the motion energy around a point (the target). Figure 2 shows a PMM in the image stream and the rectilinear representation we use to display it. In the rectilinear representation of a PMM, the horizontal axis represents the angle around the widget quantized into n_A units. The vertical axis represents the

distance from the center of the target in pixels. As will be seen, this is an excellent representation to detect and analyze the motion of objects approaching a target.

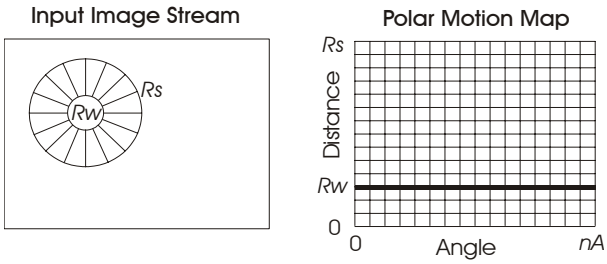

Figure 2: PMM in an image, and the rectilinear representation we use to display it.

The PMM is populated by sampling the motion mask between an inner and outer distance from the target center. Each moving pixel in that range is mapped into the PMM to turn on the corresponding pixel there. The inner radius R_w is considered the "size" of the target. In touch detection, it is the region the user must touch to trigger the event. The size of the outer radius can vary depending on the size of the widget and the expected size in the image of the object that will be touching it. This representation has similarities to the log-polar mapping found on the human retina and explored in much academic research (Traver & Pla, 2002), however we are not aware of a similar representation being used for local event detection as PMMs are here.

The Polar Touch Button (PTB) is a widget based on the PMM intended to detect when the user touches the physical object over which this widget is placed. With a single camera it is very difficult to detect when something actually touches a surface without very careful camera placement. However, there are almost always common visible elements when people are asked to touch an object. Specifically there is a relatively long narrow object that approaches the target from a consistent direction, does not pass through the target, then retracts away in the same direction in which it approached. A PTB is designed to detect these features using the PMM surrounding a target.

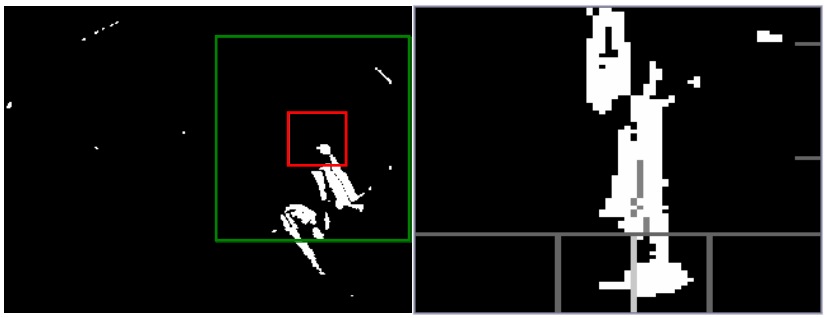

Figure 3: Motion Mask and PMM of touch (lightning strike)

The intial clue a PTB looks for to detect a touch is a distinctive motion pattern we call a "lightning strike". A lightning strike is a focused beam of motion energy that in one frame extends from the periphery (R_s) to the target (R_w) in the PMM (figure 3). After a strike is detected in one frame, it is verified by ensuring that there is sufficient motion energy focused in the same direction preceding and following it in time. The PMM representation not only makes it easy to detect and verify lightning strikes, it highlights important near-miss events such as the motion energy extending through the target to appear on the other side, as would occur if the moving object passed through the target on its way to somewhere else (figure 4). The detection algorithms have clauses designed to ignore this pattern. Gross activity, such a user walking in front of a target, creates a PMM without a distinct lightning shape, and so is easily ignored (figure 5).

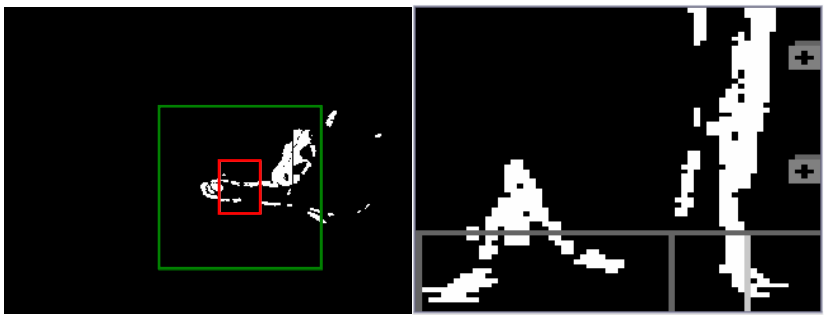

Figure 4: MM and PMM of finger extending through target

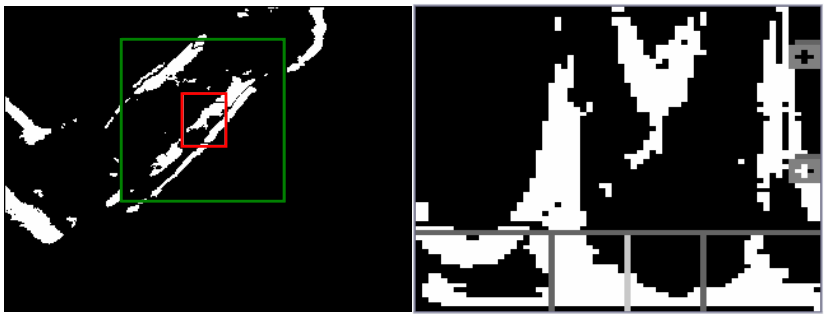

Figure 5: MM and PMM of an arm reaching through a target

Some amount of variability is needed in the detection process to encompass different pointing styles, such as a tap where a finger moves sideways into the target, then out again, as well as a straight entry, where the finger would enter the target along its axis. Some amount of flexibility is also important compensate for gaps where the motion data is sparse, as well as bursts of noise where, say, the touched surface vibrated slightly, making for a burst of motion energy. This flexibility is provided by ranges and soft thresholds in the detection routines. A paper currently in preparation will describe the touch detection algorithm in more detail.

Figure 7: One of the interactions used to evaluate PTBs

Another simple, but suprisingly useful widget built on PMMs is the Polar Track Area (PTA). A PTA returns to the application the location and distance of the motion energy nearest the target. A PTA can be used as a basic presence detector, or can be used to provide control of several continuous variables simultaneously. The distance, orientation, speed and acceleration of approach to a target can be used alone or in concert to provide 1, 2 or 3D signals which can be used to control tasks such as scrolling and navigation.

The performance of PTB was evaluated using camera data archived during a demo using earlier touch detection widgets at SIGGRAPH '01. EDVision was run on this data, replacing the previous button implementation with PTBs (figure 7). The conditions were not ideal, due to poor video quality and considering that the application had been tuned for the idiosyncracies of the previous touch detection algorithm. Even under these conditions, PTBs accurately detected 87% of the 472 touch events performed by 86 visitors to the booth. A more complete discussion of these results will be reported in the forthcoming paper.

2.3 Surfaces and Calibration

An application needs to be able to define the behaviour of widgets and their layout with respect to each other and the physical environment, as that information is relevant to the user experience. It should not be concerned with details of the recognition process, such as where widgets lie in the video image or how they are distorted by camera geometry. To provide this abstraction EDVision uses the concept of interaction *surfaces*. A surface is essentially the camera's view of a physical surface where an interface can be placed (Figure 8).

Applications refer to surfaces only by name, but each surface must be calibrated to determine its parameters, most importantly where it lies in the video image. Calibration consists of identifying four points in the image coordinate system that correspond to the corners of a rectangle in the configuration coordinate system. When a system is installed in an environment, each surface which will be used by an application is calibrated by locating these points either manually or automatically. Manual calibration consists of showing the camera's video stream on the screen and having a human identify the calibration points in the image.

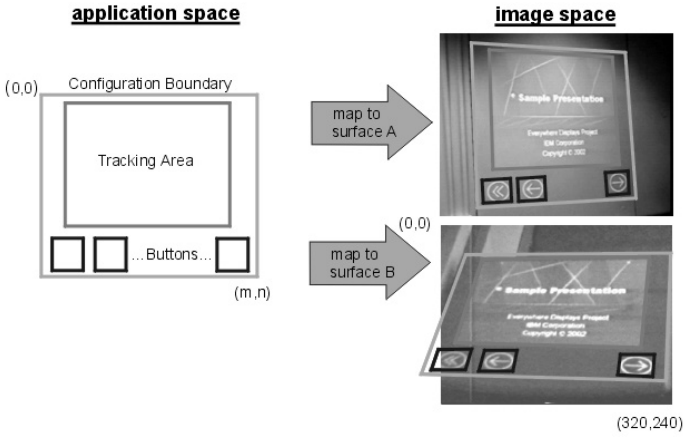

Figure 8. Mapping a configuration onto two different surfaces.

Automatic calibration consists of identifying the calibration points on the surface itself, either by placing unique markers on the surface, or in the case of a projection system, by flashing projected calibration points on the surface.

When the system is in operation, an application defines a widget configuration and names a surface where it should appear. At this point, the corners of the calibration rectangle in the configuration and the surface calibration points form 4 point pairs, which are used to compute a homography that converts between the configuration's coordinate system and image coordinate system using the methods described in (Wolberg 1994) pages 52-56.

2.4 Control

To give the application control of a VB-UI, we have defined an API and implemented it as a dialect of XML we call VIML (for Vision Interface Markup Language). VIML defines a set of visual interface objects and methods corresponding to the concepts we have discussed above. The three basic objects are: *VIsurface* for defining attributes of a surface; *VIconfiguration* for defining widgets, their spatial relationships and elaborating their behaviour; and *VIevent* for communicating events, such as a button press back to the application. In this paper we are concerned only with the three primary methods for VIconfigurations and VIsurfaces: "Set", used for setting values of objects; and "Activate/Deactivate" which control when they are operational.

"Set" commands can be issued to adjust the external parameters of objects, e.g. the location and size of a button, the resolution of a tracking area, etc. Once an object has been configured with "Set", it can be started and stopped as needed with "Activate" and "Deactivate" commands. Once activated, visual interface widgets begin to monitor the video stream and return relevant events to the application.

When a widget detects a user interaction, it returns a VIML event that identifies the event type, configuration and widget by name. VIML events are XML valid strings that can be parsed by the application. These events are interpreted and handled by the application to control the flow of execution.

The full syntax of VIML, which includes other objects and methods, is beyond the scope of this paper, but is available in a separate publication.

2.5 Example Application: Interactive Web Page

We were recently asked to create a VB-UI for a series of projected web pages. The web pages consist of a series of questions which guided a user through the process of selecting a digital camera (figure 9). Each page contains information and several questions with response buttons and an occasional scroll bar. The solution needed to be light-weight and easy to adapt to several similar, but non-identical web pages.

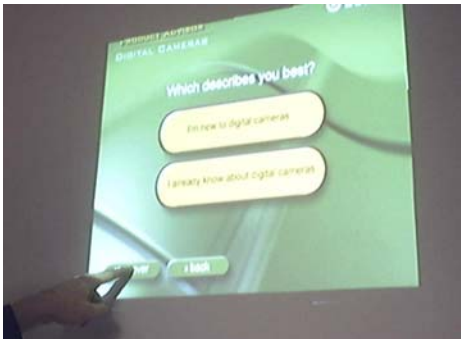

Figure 9: Interactive web page

Our solution was to use a standard web browser to display each page on a secondary screen, which then was projected to the target location. We developed a JavaScript program which could be placed in the header of each web page, and called when the page was displayed. This program would encode the widget configuration for that page in VIML, and send it to a separate EDVision application. The widgets were configured to generate mouse events on the computer screen at a location corresponding to their configuration coordinates. By calibrating the surface such that the coordinate systems of the configuration and the screen coincided, a widget placed over a projected web link would then generate a mouse click on the link, causing the browser displaying the web page to take the correct action, generally opening a new web page with its own widgets, and so on. A fingertip tracking slider widget would generate the appropriate mouse events to control the slider it was located over.

This approach has several advantages. Because the JavaScript on every page is identical, a page can be made interactive in just a few minutes by modifying a code template to create the specific widgets for that page. Because

the JavaScript gets executed automatically when a web page is displayed, and the widgets trigger the appropriate actions automatically by way of the mouse events they generated, no additional control structure is needed. The system has been deployed in several trade show venues, where hundreds of untrained users interact with the system, and has performed very well in nearly every case.

The situations where interaction performance is not up to par generally fall into one of two circumstances, both having to do with the unusual lighting conditions often found at exhibitions. To increase the dramatic effect of the exhibits, the ambient light level in the room is often kept low, and each exhibit is lit with bright local light. This high contrast can confuse the gain controls on the camera, making for an image with poor contrast (figure 10). So little visual information is available that there is not much that can be done to the vision-system to correct this situation, short of automatic detection and control of camera gain. Lacking that, we are limited to adjusting the lighting, adjusting the view of the camera so it includes only one brightness level and so the camera AGC (automatic gain control) can compensate, or turning off the AGC and adjusting the camera settings manually. Each of these is a time consuming trial and error process.

Figure 10: Poor contrast video

The second difficult situation is when the focused lighting produces shadows of the user's hand and arm as they interact with the system (figure 11). The widgets interpret the shadow as either a second moving object or, combined with the true hand, a single large object. As the system is watching for a single narrow object, like a finger or arm, this can cause the touch detection algorithm to ignore the motion entirely. Here lighting can also be adjusted to reduce the shadow, but a better solution is modify the system in one of two ways. Either the motion detection can be adjusted to detect and ignore shadows based on their appearance (Prati, Mikic, Trivedi & Cucchiara, 2003), or the interaction detection algorithms can be adjusted to detect and ignore the shadows based on their motion properties. Work is continuing along these lines.

Figure 11: Strong shadows.

3 Putting the user in charge

With this modular targeted approach to vision-based user interfaces, it is possible to create an interface suited to almost any user's circumstances or limitations. The interface will not be limited to physical input devices like a keyboard or mouse, so can be placed wherever is convenient in the environment. Widgets can be placed where they are easy for the user to reach, and can take advantage of nearly any movement the user can make. In this sense it is a "flexible" user interface, free from many of the limitations of current interaction methods.

The design of the interface, however, is still in the hands of the application. Once built, the interface is static, limited by the application designer's vision and unable to adapt to unforeseen or changing circumstances. To truly leverage the flexibility of vision-based interaction, we must place the responsibility for the interface design in the hands of the user.

Why would one need this degree of flexibility in a user interface? One justification is for interfaces that will be used by people with physical disabilities. Most people in these circumstances have a unique set of abilities and needs, so is not practical to have a single interface that serves them all. Often the physical effort required to perform controlled motions is such that the user tires easily, at which point it is desirable to reconfigure the interface to enable them to use another set of movements. Even beyond this segment of the user population, however, one could easily argue that an interface which is easy to customize is something a great many of us could appreciate and take advantage of at one time or another.

We propose to allow the user more control over the flexibility inherent in VB-UIs by using visual artifacts not only to provide targets for the interactions, but also to define the interface itself. These artifacts, either physical objects, or printed or projected images, can be placed in the camera's field of few by the user. The location of the artifact determines the target for the interaction. The identity of the object determines what action is to be performed.

Figure 12: Interaction with a printed scroll bar.

The user may take a printed "scroll bar" and place it on a table near her dominant hand (figure 12). The vision system would use unique markings, or other knowledge of the objects it expects to find (in this case the line with a distinctive red color), to locate and identify the scroll bar. When the user slides their finger up and down on the scroll bar, the system will scroll the document accordingly. The printed bar could be replaced by a 3 dimensional object, possibly with moving parts, or even by a projected image of a scroll bar that the user could steer around the environment. A more complete user interface could be constructed by placing several visual artifacts around the user, each corresponding to a different control gesture or application tasks.

The idea of using physical objects as part of a user interface is nothing new. (Fitzmaurice, 1993) proposed a “Graspable” User Interface, where small blocks were laid on top of graphical objects on a horizontal screen and then used to manipulate those graphics. The blocks created physical handles for data points which the user would manipulate manually. (Ishii & Ullmer, 1997) expanded the idea as “Tangible Bits”. They envisioned a completely new style of interaction that used physical objects to represent and view data, as well as instruments, which are similar to our visible artefacts to manipulate it.

In contrast to previous systems using physical objects for interaction, we do not use the location or identity of the artifacts to control the application directly, rather they determine the gestures which are in turn used to control the application. In addition, because it relies on modular interactive gestures similar in function to the components used in current windowed interfaces, this approach avoids the need to start with a clean slate with respect to interface design.

Interestingly, there are currently products which allow just the kind of assembly of user interfaces using standard contact-based sensors. Companies (e.g. http://www.phidgets.com/) are advertising kits of physical sensors which can be easily setup and configured to provide the inputs to an application. Unfortunately the limitations of active hardware with wire interconnects make this approach cumbersome.

Clearly, it is possible to provide the user with a software package which allows them to configure interactive widgets in the environment using on-screen tools, and thus avoid the need for physical artifacts. By dragging icons around a window displaying the video stream, the user could create a custom interface. A process like this, however, is beyond the abilities of many potential users. We believe that moving visible artifacts about the environment is a far more intuitive means to the same end.

3.1 Defining an Interface with Visual Artifacts

Specifically, we propose providing the user with a set of visually unique artifacts, each being either a physical object, or a printed or projected image. A complimentary computer vision system is capable of recognizing the artifacts, detecting user movements near them, and generating computer events in response. The vision system scans the environment to find the current location of the interaction objects. It then monitors that location for the expected interaction action (touch, etc), converts the interaction into a control signal, which is then translated to an application specific event.

The appearance of each object determines the type of user interaction it recognizes and the nature of the computer event it generates in response. For example a "button" object responds with a binary event when touched by the user. A "slider" object controls the value of a one dimensional parameter corresponding to the location of the users touch. A "track area" object controls a two dimensional parameter in response to the location of the user's touch. As discussed above, objects fall into categories corresponding to the dimensionality of the interaction they support. A 0D object will generate a binary event, 1D object will generate a one dimensional parameter, and so on. True physical objects (as opposed to prints or projections) may have moving parts which may be manipulated during the interaction.

Each class may have several objects supporting different types of interaction, for example a 1D class may have sliders which control a signal by the position of the user's hand along a line, circular objects that generate their signal corresponding to the angular position of the user's fingertip along the perimeter, and proximity objects that generate a signal based on the distance of the user's hand from the target. Similarly, there may be several ways to generate a 0D, 2D or 3D events, each associated with a visually distinct object. A user may choose any object of a class to generate the control signal of that class needed by an application.

An object may have several types of events it generates in response to different actions. For example an object may generate a binary event in response to a touch and a 1D signal in response to a finger slid along it. The control signals, in turn, can be used to generate application specific events. For example a slider's 1D signal may be used to control scrolling within a document display, or may be mapped to any other 1D application specific parameter. The mapping of control signal to application action may be predefined, or determined by the user.

The user now has a natural way to structure a user interface suited to their immediate needs. Not only can the layout of the widgets be customized, but the actual interactions used to generate the control signals can be changed as well. This is done without programming or computer interaction of any kind, simply by the user moving objects around them.

This concept has a host of issues which must be resolved in order to be a practical and general purpose solution. One is the method of associating the events generated by widgets with the inputs needed by an application. For any single application it is reasonable to have a dedicated set of artifacts which map directly to the required inputs. When the user wants to switch applications, however, they should not have to reconfigure their artifacts to match. Another issue is the sheer number of inputs required by many current applications. Finally, in the current form a user MUST set up the interface every time they want to use it, and can not take advantage of a system default interface. In spite of these issues, we believe this approach has promise. We are exploring solutions such as menu widgets and using a combination of projectors and physical artifacts to create an interface that is configurable by both the user and the system. However, even in its current form, the approach of using visual objects to customize a user interface has great potential in circumstances where the user has special interface requirements.

4 Summary and Conclusion

Gesture-based user interfaces have the potential for nearly limitless flexibility in user interface design. We have argued that in some ways they provide too much flexibility, and that both the user and the system would benefit from constraints on the allowable interactions. An existing system is described that limits the actual gestures used in the interface to a concise set, each with distinct meaning. Additional constraint on the gestures is provided by making them targeted, meaning they must occur at a specific location that is obvious to the user and known to the system. Complex interfaces can be created by combining these basic gestures, much as current WIMP interfaces are assembled from on-screen widgets.

These systems, however are still limited in their flexibility because once coded, the interface can not be changed. We described a method to extend this concept of modular target-based interactions by having the system automatically detect visible artifacts which the user can select and move about the environment, and in so doing, create a custom user interface, suited to their immediate needs.

References

Davis, L., ed. (2002) Proc. of the 5th International Conference on Automatic Face and Gesture Recognition (FG 2002), IEEE Computer Society: Washington, DC.

Fitzmaurice, G. (1993). Situated Information Spaces and Spatially Aware Palmtop Computers. *Communications of the. ACM,* 36(7), 38-49.

Ishii, H., Ullmer, B., (1997) Tangible Bits: Towards Seamless Interfaces between People, Bits, and Atoms. In: Proc. of CHI'97. Atlanta, Georgia. pp. 234-241

Kjeldsen, R., and Hartman, J., (2001). Design Issues for Vision-based Computer Interaction Systems. In Proceedings of Perceptual User Interfaces,

Kjeldsen, R., Levas, A., Pinhanez, C., (2004). Dynamically Reconfigurable Vision-Based User Interfaces. *Machine Vision and Applications*, 16(1), 323

Prati A., Mikic, I., Trivedi, M., and Cucchiara, R. (2003). Detecting moving shadows: Algorithms and evaluation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25, 918–923

Traver, V., Pla, F., Motion estimation and figure-ground segmentation using log-polar images. In Proceedings of the 16th Intl Conference on Pattern Recognition, 2002

Turk, M., ed. (2001) Proc. of the Workshop on Perceptual/Perceptive User Interfaces. Orlando, Florida.

Wolberg, G., (1994) Digital Image Warping. IEEE Computer Society Press.

Wu, Y. and Huang, T., (1999) Vision-Based Gesture Recognition:A Review. *Lecture Notes in Artificial Intelligence* 1739.

Ye. G., Corso, J., Burschka, D.,, and Hager, G., (2004). Vics: A modular hci framework using spatio-temporal dynamics. *Machine Vision and Applications*, 16(1):13-20.