A Practical Approach for Integrating Vision-Based Methods into Interactive 2D/3D Applications

Darius Burschka, Guangqi Ye, Jason J. Corso, Gregory D. Hager

Technical Report CIRL-TR-05-01
Computational Interaction and Robotics Laboratory
The Johns Hopkins University

Abstract

In this paper, we present a practical approach for integrating computer vision techniques into human-computer interaction systems. Our techniques are applicable in conventional 2D windowing environments and advanced 3D scenarios. The method is transparent in the sense that no modification (e.g. no re-compilation) of the underlying application software is required. To realize this functionality, we propose an interface scanner *that analyzes the input event requirements of a given running application and translates them into event descriptions available from our vision-based HCI framework. We include a description of our vision system that extends the basic set of events like ButtonPressEvent, MotionEvent, ButtonReleaseEvent by more abstract events like TurnDialEvent, DeleteEvent.*

We describe our framework for 2D interfaces based on the Unix X11 API and for 3D interfaces which use OpenGL. We provide an analysis of the stability and accuracy of the framework for our current implementation.

1. Introduction

Present applications are designed to use conventional input devices, like keyboard or computer mouse. The limited set of the available input events has required applications to define a complex set of *mode-buttons*[1] and often, a difficult *interaction language* for the user to learn. Vision-based interfaces have the potential to provide a richer set of events that extends the basic set of typical events provided by the current graphical systems, like our target Unix X11-graphical environment. These additional events would allow the user to perform more complex tasks in a direct way through natural hand gestures and functional motions to which they are accustomed in everyday life.

The approach we present in this paper allows an easy transition to vision-based interfaces by translating the richer stream of visual events into existing event streams of typical graphical interfaces. For example, the action of pressing a button is translated into a mouse click on the button to effect the same trigger in the system. The structure of an event stream in the typical graphical Unix application relies on communication between the server program (XServer) and the client application (Figure. 1).

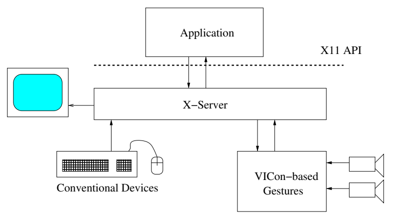

Figure 1: *Extension of the common graphical 2D API.*

The XServer provides a transparent layer allowing the passing of events between input devices and applications. Additional input devices can be added to generate events that are sent to the applications. This modular feature allows us to

[1]We define a *mode-button* as an application element that effects a change in the application mode. An example where mode-buttons are needed is in switching among the various modes of a line-drawing tool (freehand, straight-line, poly-line, etc) in a drawing program.

supplement traditional devices with our framework without any modifications to the existing client software.

Typically, human-computer interaction (HCI) systems continuously track the user in order to model his or her actions [5, 20]. However, an alternative approach has recently been suggested by two independent groups [8, 24]. Instead of globally tracking and modeling the user, these papers suggest monitoring a local volume around each interface component. The authors suggest using a hierarchical processing model such that coarse, inexpensive operations like image differencing are used to trigger more expensive operations like template matching. They use state machines to define a network of operations which recognize certain actions per volume above each interface element. We adopt this approach in this paper because it provides a concrete methodology for applying vision techniques to both conventional and experimental interfaces.

The rest of the paper is organized as follows. We provide a short survey of related research in Section 1.1. Section 2 discusses our framework of combining vision into 2D/3D interfaces. In Section 3, we explain the details of our current 2D and 3D implementation of the proposed framework. Section 4 presents some experimental results of our implementation. Finally, we provide the conclusions and a short discussion.

1.1. Related Work

Generally, vision-based interaction is based on recognizing human body motion and facial expression with the general goal of supporting HCI [2, 4, 11, 12, 22]. Many interaction systems [9, 10, 15, 16, 14, 18, 25] have been implemented to show the promise of vision-based HCI.

Hand gestures have been a popular choice in vision-based HCI because of their convenience and naturalness. Most gesture systems employ a tracking-based approach to recognize a small set of gestures. Wilson and Oliver [20] presented the “GWindows” system that combines vision and speech to augment traditional “WIMP” (windows, icons, mouse, pointer) interfaces. Both motion tracking and depth computation are used to recognize pointing and moving gestures. One of the drawbacks of the system is the limited gesture set which does not contain a selecting gesture.

A bare-hand HCI framework is proposed in [5]. The hands are segmented from the background via image differencing. The fingertips of the user’s hands are detected and grouped. Gestures are defined as different finger configurations. Zhang et al. [25] present a “visual panel” system that uses a single fingertip as the sole interaction medium. The lack of a temporal model for dynamic gestures is compensated by a simple timer, i.e., a certain gesture command is issued when the user stays in the same configuration for a predefined period of time. The SocialDesk system [9] uses a “shadow infrared scheme” to segment the user’s hand from the background, which is a projected desktop. They model gesture commands as a transition between two postures. One limitation is the 2D nature of the system, which can not handle 3D gestures.

Vision-based HCI has also been incorporated into virtual environments and augmented reality systems [11, 17, 18, 23]. Both manipulative [1, 21] and communicative gestures [18] are used to interact with virtual or real objects. The Perceptive Workbench [18] system uses multiple infrared light sources and a gray-scale camera with an infrared filter to detect the 3D shape of the hand and the objects in the system. Gestures are recognized based on tracked hand position and pointing direction. Several applications, such as terrain navigation and virtual game playing, have been built using the system. However, the gesture vocabulary is limited compared to the complexity of the system and the capability of 3D construction.

2. Localization of Interface Elements

As discussed in Section 1.1, extensive research has been done in the area of human tracking for HCI. Many approaches track the user and focus on the analysis of the human posture and motion. The sequence of gestures is generated relative to the event source, in this case the user. This requires the system to recognize all possible gestures at all locations in the interface. However, since most interface components occupy a small area of the interface and respond to only a subset of the gestures, this will result in unnecessary processing. Recently, an alternative approach was suggested by [8, 24] to address this problem.

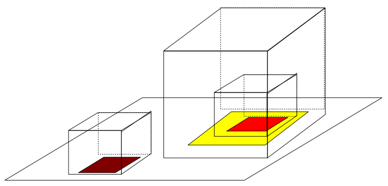

Figure 2: *Each element of the interface defines its own volume of interest in space. The volumes can contain each other in case of hierarchical element structure.*

In this approach, each interface element observes a volume in space and defines a set of gestures to which it responds.

Any user activity detected in this volume is forwarded to this element (Figure. 2). These volumes can contain each other in case that a graphical interface element consists of a hierarchy of regions with a parent region and embedded sub-regions. We developed parsers that use existing interface definitions for 2D and 3D applications to define the positions of the volumes in space. In both, the 2D and 3D cases, the vision component localization is derived from the graphical language used to generate the visual output on the screen. Therefore, the vision-based gesture recognition is tailored to the requirements of the application.

2.1. 2D Case

Conventional programs. The basic calculator application on X11-based systems, the *xcalc* (Figure. 3), is an example that demonstrates the hierarchical structure depicted in Figure 2. The entire calculator area can be used for detection of the hand in the volume above it, and the single keys of the calculator carve sub-volumes in this regions monitoring for KeyPress events. To avoid the modification (i.e. re-compilation) of existing client software, the event requirements for a given application are parsed directly from the XServer screen structure.

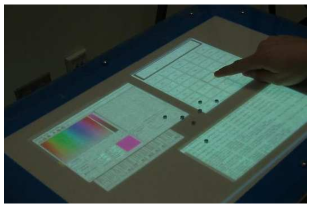

Figure 3: *The X11 calculator application opens separate subwindows for each of the keys specifying the type of event that activates it.*

The XServer defines a nested hierarchy of windows (Figure 4). All application windows are children of the RootWindow which represents the entire screen. Using the *XQueryTree()*, a complete window hierarchy can be retrieved based on the application name.

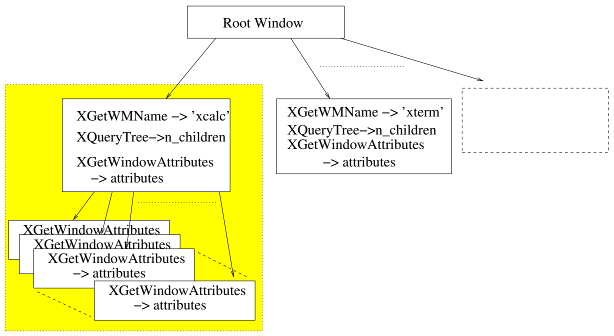

Figure 4: *Application windows and requested events can be identified in the screen hierarchy starting at the RootWindow that represents the entire screen.*

Once the corresponding window structure is found, the HCI system identifies the requested event types for this window using the *XGetWindowAttributes()* call. The events are mapped onto the user gestures in our HCI Framework (Section 3.1.1). Typical X11 application gestures are: ButtonPressed, ButtonReleased, MotionNotify accompanied by the x,y coordinates of the current position of the screen. In Section 3.1.1, we explain the specific implementation we have used for this mapping.

The vision-based system passes these events to the applications using the identified window structures using the *XSendEvent()* call.

Extensions As we have mentioned in the introduction, human gestures provide a more powerful set of input events than conventional input devices. Additional event types can be generated by different gestures presented to the system. Possible examples are DestroyEvent, TurnDialEvent, etc. These events can be added to the current XServer definition or masqueraded as ClientEvents, which are part of the default XServer definition.

2.2. 3D Case

In analogy to Section 2.1, we need to define the position of the active volume for the gesture recognition in front of the surface of an interface element in 3D space. The simplified situation of a planar interface geometry in 2D with its well-defined graphical definitions for interfaces in applications is not given in 3D. However, 3D interfaces are still not common and they are not standardized. In our framework, we use OpenGL world models to define surfaces that accept user interaction. In our implementation, we use GL_QUADS to model the surfaces (vertices) of our virtual objects in space. The rendered scene is overlayed with the regular stereo camera view.

The actual processing of gestures and creation of the video-based events proceeds similarly to the 2D case with the only difference that the surface orientation is now arbitrary in space. It is completely defined by the quad definitions in the OpenGL model that is used for rendering.

3. Vision-Based Events

In Section 2, we proposed an approach for the localization of vision-based interface components based on a given graphical interface. Here, we present our implementation of the vision-based event generation.

3.1. 2D Interfaces

The control of most conventional 2D interfaces has been limited to the keyboard and mouse for practical purposes. This has led to a fairly small set of interface elements and associated events: e.g. button, scrollbar, menu (which is a button), etc. In this section, we describe our techniques for 2D interfaces which map gestures to both these existing events and to new events. Our experimental platform is a copy of the touchpad system presented in [3], but we have enhanced their setup by lifting the homographic calibration to full 3D calibration and also adding color calibration between the display and the cameras.

3.1.1 Mapping Events

In this section, we describe the specific translations from vision-based gesture events to conventional events for graphical applications. As discussed earlier, for each interactive element of a program, we automatically (and dynamically) create a vision interactive element which looks for the expected human gesture input. For conventional input, the possible interactive elements are limited to clickable items and draggable items. Thus, we must only define two distinct mappings to encompass all conventional application control; this is expected since the mouse is very simple controlling device.

For clickable items, we define a natural pressing gesture that is performed by the user extending one outstretched finger near the interactive element, touching the element and then removing the finger. A ButtonPress event is generated when the visual parser observes the finger touching the element and then a ButtonRelease event when the fingers moves away.

For draggable items a natural gesture is again used: the user approaches the item with two opposing fingers open, and upon reaching the item, he or she closes the fingers as if grasping the item (there is no haptic feedback). Then, the user is permitted to drag the item around the workspace, and whenever he or she wishes to drop the item, the two grasping fingers are released quickly. The corresponding events we generate are ButtonPress, MotionNotify, and ButtonRelease.

Figure 5 shows examples of the two gestures from one camera. The accuracy of the network is shown in Table 1 for pressing and Table 2 for grabbing.

3.1.2 Feature Extraction

Robust hand detection is a crucial prerequisite for gesture capturing and analysis. Human hands demonstrate distinct appearance characteristics, such as color, hand contour, and geometric properties of the fingers. We carry out segmentation based on the appearance of the hand.

Assume that the background is known[2], the hand can be detected by segmenting the foreground from the image. The key is to find an efficient and robust method to model the appearance of the background and the hand. Background subtraction, gray-scale background modeling [6], color appearance modeling [19], color histogram [7] and combining of multiple cues [21] are among the most widely used methods to model the background and perform foreground segmentation.

We propose to directly model the appearance of the background by computing the transform between the physical scene and the images of the scene captured by the cameras. By this means, when the user's hand enters the scene, we can detect the hand by performing foreground segmentation based on the modeled background.

[2]In the 2D touchpad setup this is trivial since the background is precisely what is rendered on the screen at any given moment.

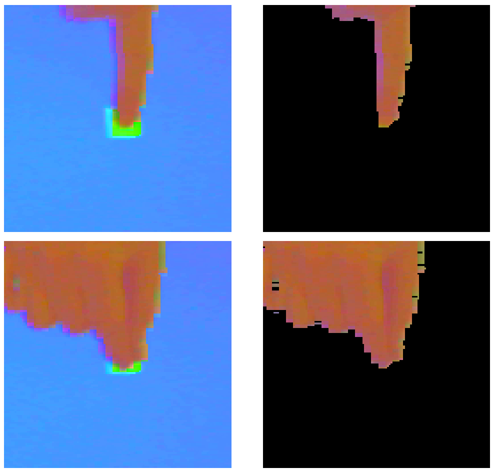

Figure 5: *(Top) Pressing Gesture. (Bottom) Grabbing Gesture. (Left) Original Image. (Right) Segmented Image.*

We model the color appearance of the background using an affine model. The color images from the cameras and the rendered scene are represented in YUV format. An affine model represents the transform from the color of the rendered scene, i.e., $s = [Y_s, U_s, V_s]^T$, to that of the camera image $c = [Y_c, U_c, V_c]^T$. Using a 3×3 matrix A and a vector t, we represent this model using the following equation.

$$c = As + t \tag{1}$$

The model parameters, A and t, are learned via a color calibration procedure. Basically, we generate a set of N scene patterns of uniform color, $\mathcal{P} \doteq \{P_1 \dots P_N\}$. To ensure the accuracy of the modeling over the whole color space, the colors of the set of the calibration patterns occupy as much of the color space as possible. We display each pattern P_i and capture an image sequence S_i of the scene. The corresponding image C_i is computed as the average of all the images in the sequence S_i. The smoothing process is intended to reduce the imaging noise. For each pair of P_i and C_i, we randomly select M pairs of points from the scene and the images. We construct a linear equation based on these $N \times M$ correspondences and obtain a least squares solution for the 12 model parameters.

We use image differencing to segment the foreground. Given background image I_B and an input image I_F, a simple way to segment the foreground is to subtract I_B from I_F. We compute the sum of absolute differences (SAD) for the color channels of each pixel. If the SAD is above a certain threshold, this pixel is set to foreground. Figure 6 shows an example of the segmentation.

To improve the robustness of the foreground segmentation, we include an additional skin color model. Many skin models have been proposed in the literature [13]. Here we choose a simple linear model in UV-space. Basically, we collect skin pixels from segmented hand images and train the model as a rectangle in the UV plane. Four parameters, i.e., $U_{min}, U_{max}, V_{min}, V_{max}$, are computed and used to classify image pixels.

3.1.3 Gesture Modeling

For the gesture mapping discussed in this paper, we use a set of neural networks to perform the actual recognition. As discussed earlier, since an event window is monitored in the images for each of the interface components and no user tracking is required for the gesture recognition in this form, a neural network is a suitable technique to perform the recognition of the gesture posture in the image window. We train a standard three-layer binary network for recognition. We choose a coarse sub-sampling of the segmented image window as input to the network. We use the normalized intensity to reduce photometric distortion.

3.2. 3D Interfaces

In this section, we discuss our approach for incorporating vision based analysis of interaction in 3D settings. In 2D, we assume the interactive regions of interest lie on a plane in some given distance and orientation. However, in 3D, we make no such assumption. The target system consists of a Head Mounted Display (HMD) with two small analog cameras mounted on

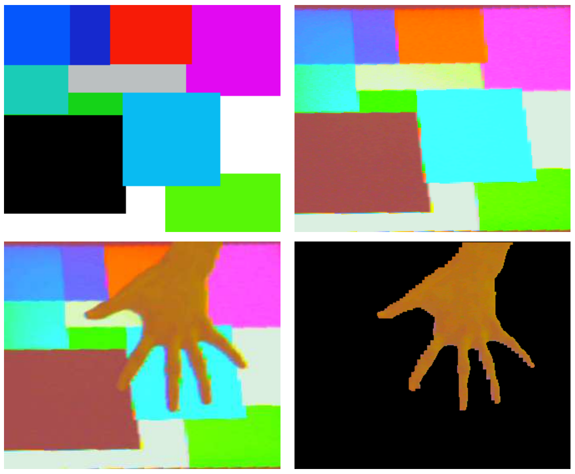

Figure 6: *An example of image segmentation based on color calibration. The upper left image is the the original pattern rendered on the screen. The upper right image is the geometrically and chromatically transformed image of the rendered pattern. The lower left image shows the image actually captured by the camera. The lower right image is the segmented foreground image.*

the sides providing the video. In our current experiments, we use a regular stereo camera rig and the result is viewed on the desktop monitor. A setup with a virtual box rendered in space is depicted in Figure 7.

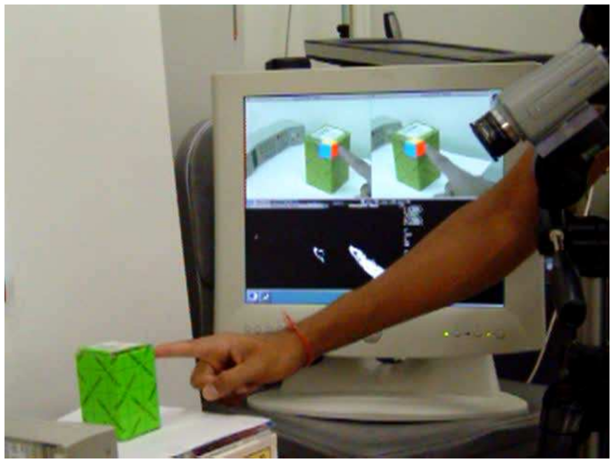

Figure 7: *Setup for the 3D camera experiment. A physical box is included only for verifying the accuracy.*

3.2.1 Assumptions

Before we can analyze a 3D interactive application, its GUI must be defined in a virtual 3D environment. Furthermore, this GUI must be accessible to our analysis tool which will add a buffer volume around each interactive region. Finally, the cameras used to capture the human interaction should be properly calibrated.

3.2.2 Interface Component Visibility

In this development, we assume that the 3D scene will be rendered based on the camera calibration. We also assume that the active part of the 3D application will be in the viewing frustum of the cameras. We check if each interactive region can be seen by the camera system by taking the dot-product of the camera optical axis and the normal of the surface. If the dot-product is negative, the surface is visible to the camera. If the surface is visible to both cameras, the interface element is considered active.

	Training	Testing
Positive	$2051/2051$	$2057/2058$
Negative	$1342/1342$	$1342/1342$

Table 1: *Recognition Accuracy for Pressing Gesture.*

	Training	Testing
Positive	$366/367$	$364/367$
Negative	$480/480$	$483/483$

Table 2: *Recognition Accuracy for Grabbing Gesture.*

3.2.3 Video Processing

The 3D interface elements act as a transparent layer to the actual interactive 3D region below. For each visible region, we perform coarse image processing (e.g. motion detection) on the projection of the volume seen above the visible surface. The region undergoes finer image processing as specified in a predefined state machine. Using two or more cameras, the disparity of points within the volume can be computed and used to extract more information about the change within the region.

4. Experimental Results

4.1. 2D Segmentation

We have carried out a series of experiments to quantify the accuracy and stability of the system. First, we examine the robustness and stability of the color calibration algorithm. We train the affine color model using 343 unicolor image patterns which are evenly distributed in the RGB color space. To test the accuracy of the learned affine model, we display over 100 randomly generated color images and examine the resulting segmentation. In this case, the ground truth is an image marked completely as background pixels. For both cameras, the system achieves segmentation accuracy of over 98%.

We also investigate the efficacy of the linear skin model. We learn the model by analyzing image sequences containing the hands of over 10 people. To test the model on our platform, the user is asked to place his or her hand on the flat-panel and keep it still. Next, we render a background which is known to perform well for skin segmentation and treat the resulting skin foreground segmentation as the ground truth. The user is asked to keep his or her hand steady while we render a sequence of 200 randomly generated patterns. For each image, we count the number of incorrectly segmented pixels against the true segmentation. The overall skin segmentation accuracy is over 93%.

4.2. 2D Recognition Experiments

One of our example 2D mappings is to use a button press gesture to mimic a mouse click event. As discussed earlier, we train a 3-layer neural network to recognize the button press gesture and grabbing gesture. For each of these networks, there are 512 input nodes ($16 \times 16 \times 2$ for each stereo pair), 20 middle layer nodes, and 1 output node. The accuracy of the network is shown in Table 1 for pressing and Table 2 for grabbing.

We carry out an experiment to test the accuracy and spatial sensitivity of the button press gesture. We display an array of square buttons, which are adjacent to each other with no buffer-space. Then, the system randomly chooses one of them and instructs the user to press it. We vary the size of the button from 20×20 pixels to 75×75 pixels and repeat the experiment for each size. Figure 8 shows the scene when the size of the button is 40×40 pixels; here, we see that the size of the user's finger tip is about 40 pixels wide at the display resolution. Figure 9 shows the testing results. In the graph, we see the accuracy of the press recognition rapidly increases with the size the button. We also note that, as expected, for button sizes smaller than the fingertip resolution (about 40 pixels), the accuracy is much worse.

Figure 8 also shows an application using the button press gesture to control a calculator program in X window system.

4.3. 2D Pointing Stability

To investigate the stability of the location recognition and the underlying segmentation, we perform an experiment in which the user is asked to point and stay at an arbitrarily specified position on the screen. For this experiment, we have trained a neural network system to localize the spatial coordinates of the fingertip. Thus, for a given image region defined for an interface element, the images are processed by the network which yields a single (x, y) sub-pixel resolution tip-location. Again, the input to the network is a coarse, sub-sampled image (256 total samples), and the network has 16 hidden nodes.

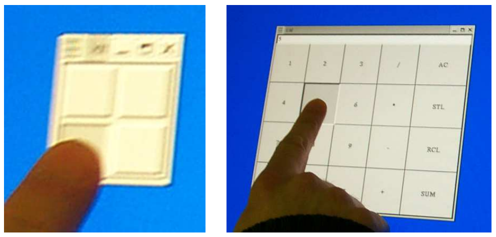

Figure 8: *Left picture shows the scene of when the finger is trying to trigger a button of* 40×40 *pixels. Right picture shows a user is controlling an X window program using finger press gestures.*

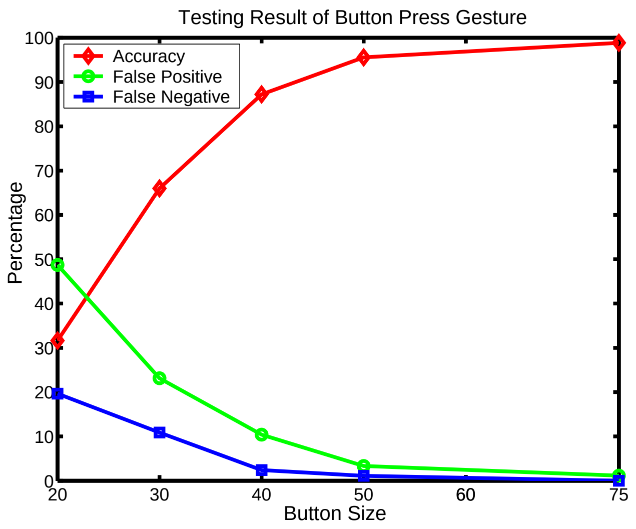

Figure 9: *2D button press experiment result*

We hand-annotated each training datum by clicking on the fingertip location. Over 600 training samples are used and the average error for the training data is 0.0099 image pixels.

To test the stability, we dynamically change the scene by rendering randomly generated background images and record the output of the pointing gesture recognizer. The stability of the system is measured by the standard deviation of the recorded 2D position. On a standard flatpanel monitor with a resolution of 1280×1024 pixels, our system reports an average standard deviation of 0.128 and 0.168 pixel in the x and y direction, respectively.

4.4. 3D Examples

The 3D interface works with user-dependent skin calibration. The system learns the skin hue and distribution from an initial sample presented in a window (Figure. 10).

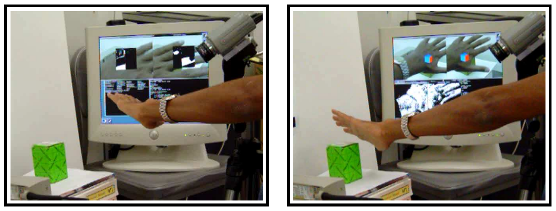

Figure 10: *(left) Initial hue calibration for user-dependent skin detection; (right) hand segmentation test.*

The calibrated system is used to extract the finger shape in both camera images. We estimate the finger tip position in both images using PCA, and reconstruct its 3D position (Figure. 11).

The 3D position of the finger is calculated relative to the active (visible) surfaces of our model. The virtual surfaces change color on contact providing visual feedback to replace the missing haptic feedback (Figure. 12). We align the cardboard box

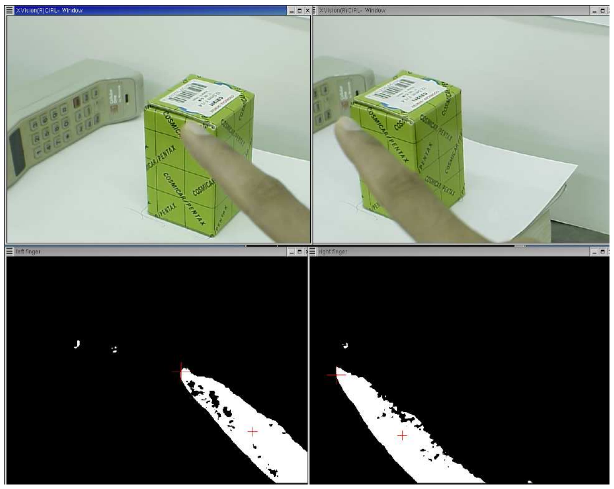
Figure 11: *Extraction of the finger tip position in both images for the 3D reconstruction.*

with the virtual box because in the current setup, the result is viewed on a regular computer monitor which does not create any 3D perception to the user.

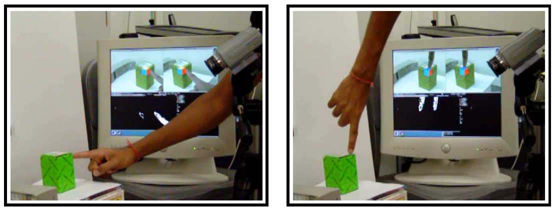

Figure 12: *Examples of 3D surface contact.*

5. Conclusions and Future Work

In this paper, we present a practical approach to integrate principles of local-based gesture modeling into both conventional and experimental 3D interfaces. The first contribution is to directly use the graphical interface descriptions to define the active area and type of expected events in 2D and 3D space. The suggested parsing of the underlying models that define the graphical layout allows an easy integration with existing client software. The second contribution is the framework for mapping visual recognition events to conventional interface events and providing a method to extend the event vocabulary.

While we have demonstrated that the complete set of 2D interface events can be mapped, there is no standard 3D interface model or event vocabulary. In our current work, we are trying to establish such an interface model. We note the similarity in our approaches to visual recognition in 2D and 3D: essentially, the methods reduce to analyzing image regions for expected visual events. Thus, the main difference between the 2D and 3D case as presented in this paper is the method for foreground segmentation and feature extraction. Therefore, we expect to merge the 2D and 3D gesture recognition in future work.

References

[1] D.A. Becker and A.P. Pentland. Using a Virtual Environment to Teach Cancer Patients T'ai Chi, Relaxation, and Self-Imagery. In *Proc. International Conference on Automatic Face and Gesture Recognition*, 1996.

[2] M.J. Black and Y. Yacoob. Tracking and Recognizing Rigid and Non-rigid Facial Motions Using Local Parametric Models of Image Motion. *International Journal of Computer Vision*, 25(1):23–48, 1997.

[3] J.J. Corso, D. Burschka, and G.D. Hager. The 4DT: Unencumbered HCI With VICs. In *CVPR HCI*, 2003.

[4] A. Elgammal, V. Shet, Y. Yacoob, and L.S. Davis. Learning Dynamics for Exemplar-based Gesture Recognition. In *Computer Vision and Pattern Recognition*, volume 1, pages 571–578, 2003.

[5] C.v. Hardenberg and F. Berard. Bare-Hand Human-Computer Interaction. In *Workshop on Perceptive User Interfaces*, 2001.

[6] T. Horprasert, D. Harwood, and L.S. Davis. A Robust Background Substraction and Shadow Detection. In *Asian Conference on Computer Vision*, 2000.

[7] M.J. Jones and J. Rehg. Statistical Color Models with Application to Skin Detection. *International Journal of Computer Vision*, 46(1):81–96, 2002.

[8] R. Kjeldsen, A. Levas, and C. Pinhanez. Dynamically Reconfigurable Vision-Based User Interfaces. In *International Conference on Computer Vision Systems*, pages 323–332, 2003.

[9] C. McDonald, G. Roth, and S. Marsh. Red-Handed: Collaborative Gesture Interaction with a Projection Table. In *Proc. International Conference on Automatic Face and Gesture Recognition*, pages 773–778, 2004.

[10] T. Moran, E. Saund, W. van Melle, A. Gujar, K. Fishkin, and B. Harrison. Design and Technology for Collaborage: Collaborative Collages of Information on Physical Walls. In *ACM Symposium on User Interface Software an Technology*, 1999.

[11] K. Oka, Y. Sato, and H. Koike. Real-Time Fingertip Tracking and Gesture Recognition. *IEEE Computer Graphics and Applications*, 22(6):64–71, 2002.

[12] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997.

[13] S.L. Phung, A. Bouzerdoum, and D. Chai. Skin Segmentation Using Color Pixel Classificatioin: Analyssi and Comparision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(1):148–154, 2005.

[14] J. Segen and S. Kumar. Fast and Accurate 3D Gesture Recognition Interface. In *International Conference on Pattern Recognition*, 1998.

[15] J. Segen and S. Kumar. Gesture VR: Vision-based 3D Hand Interace for Spatial Interaction. In *ACM International Conference on Multimedia*, pages 455–464, 1998.

[16] J. Segen and S. Kumar. Shadow Gestures: 3D Hand Pose Estimation Using a Single Camera. In *Computer Vision and Pattern Recognition*, volume 1, pages 479–485, 1999.

[17] J.M. Siskind. Visual Event Perception. In *Proceedings of the NEC Research Symposium*, 1998.

[18] T. Starner, B. Leibe, D. Minnen, T. Westyn, A. Hurst, and J. Weeks. The Perceptive Workbench: Computer-Visioin-Based Gesture Tracking, Object Tracking, and 3D Reconstruction for Augmented Desks. *Machine Vision and Applications*, 14(1):59–71, 2003.

[19] M.J. Swain. Color Indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[20] A. Wilson and N. Oliver. GWindows: Robust Stereo Vision for Gesture-Based . In *Workshop on Perceptive User Interfaces*, pages 211–218, 2003.

[21] C. Wren, A. Azarbayejani, T. Darrell, and A.P. Pentland. Pfinder: Real-time Tracking of the Human Body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–784, 1997.

[22] Y. Wu, J.Y. Lin, and T.S. Huang. Capturing Natural Hand Articulation. In *Proc. Int'l Conf. Computer Vision*, volume 2, pages 426–432, 2001.

[23] G. Ye, J.J. Corso, and G.D. Hager. Gesture Recognition Using 3D Appearance and Motion Features. In *Proceedings of CVPR Workshop on Real-Time Vision for Human-Computer Interaction*, 2004.

[24] G. Ye, J.J. Corso, and G.D. Hager. VICs: A Modular HCI Framework Using Spatio-Temporal Dynamics. *Machine Vision and Applications*, 2004.

[25] Z. Zhang, Y. Wu, Y. Shan, and S. Shafer. Visual Panel: Virtual Mouse Keyboard and 3D Controller with an Ordinary Piece of Paper. In *Workshop on Perceptive User Interfaces*, 2001.