User-Centric Design of a Vision System for Interactive Applications

Stanislaw Borkowski[†], Julien Letessier[†§], François Bérard[§], and James L. Crowley[†]

[†]INRIA Rhône-alpes
GRAVIR laboratory – PRIMA group
655 avenue de l'Europe – Montbonnot
38334 Saint Ismier Cedex – France
{stan.borkowski,julien.letessier,james.crowley}@inrialpes.fr

[§]CLIPS – IMAG
HCI group
BP 53
38041 Grenoble Cedex 9 – France
francois.berard@imag.fr

Abstract

Despite great promise of vision-based user interfaces, commercial employment of such systems remains marginal. Most vision-based interactive systems are one-time, "proof of concept" prototypes that demonstrate the interest of a particular image treatment applied to interaction. In general, vision systems require parameter tuning, both during setup and at runtime, and are thus difficult to handle by non-experts in computer vision. In this paper, we present a pragmatic, developer-centric, service-oriented framework for the construction of vision-based interactive systems. Our framework is designed to allow developers unfamiliar with vision to use computer vision as an interaction modality. To achieve this goal, we address specific developer- and interaction-centric requirements during the design of our system. We validate our approach with an implementation of standard GUI widgets (buttons and sliders) based on computer vision.

1. Introduction

Computer vision has the potential to provide a rich modality for user input in interactive systems. Numerous examples from the literature illustrate the diversity of application domains and interaction styles created using vision as input modality [7, 11, 9, 14, 2, 12, 10]. Yet, there are surprisingly few examples of commercially available vision-based interactive systems.

Human-computer interface (HCI) developers are used to hardware-based input devices that provide data "out of the box" with little or no off-line or runtime tuning. In contrast, vision-based input systems can only be integrated by HCI developers with great difficulty: they require vision expertise for setup, as well as execution conditions incompatible with typical HCI requirements. In this sense, most current computer vision systems do not *just work*.

We believe that to popularize vision-based interfaces, computer vision systems must be made more developer-friendly as well as more user-friendly. To accomplish this, it is necessary that vision system developers adopt a different perspective. In this paper, we present an approach to vision system design that aims at minimizing the difficulties related to the deployment of vision-based interactive systems by: *(a)* encapsulating vision components in isolated services, *(b)* imposing these services to meet specific usability requirements, and *(c)* limiting communications between the services and the interactive applications to a minimum. We describe our approach in section 3.

In section 4, we design a system for vision-based interactive widgets for augmented surfaces, and propose a breakdown into atomic services. In this context, we identify the necessary information exchange between services and with the application. Meeting the requirements detailed in section 3 influences the choices of the image processing algorithms used; in particular, they require little or no setup and tuning and feature usability-grade latency. Our implementation is robust to light changes and is moderate in CPU usage. Finally, in section 5 we detail an implementation of a simple vision-based calculator built using the described vision system.

In this paper, we consider the requirements of developers and end-users. By "developers", we refer to the developers of the interactive systems aimed at "end-users". In particular, these are not the developers of the vision system and we do not expect them to have computer vision expertise. By "end-users", we refer to the users of the final system, who are assumed to have neither developer nor computer vision expertise.

2. Related work

Much work has been done in vision-based gesture recognition and novel interaction styles research, but the problem of integrating vision to standard development has only recently received attention. Kjeldsen et al.[5] present an architecture for dynamically reconfigurable vision-based user interfaces (VB-UIs). They propose to separate the application from the vision engine. The core application controls the vision engine by sending XML messages specifying the current interface configuration. The interface is composed of interaction widgets defined at the level of functional core by their function and position in the interface. As end-users actuate widgets, the image processing engine sends messages that correspond to interaction events to the application's functional core.

The modular decomposition of vision components, as well as the separation of image treatment from the functional core of applications are important steps toward "developer-usable" VB-UI components. However, as presented, the VB-UI approach requires its developer to define sets of parameters and calibration data for the vision treatment engine for each interface configuration and location. Setting parameters for particular conditions allows fine tuning of the vision system. On the other hand, it requires the user of the vision system (the developer) to understand the underlying image processing algorithms, which is contradictory with the idea of VB-UI deployment by developers unfamiliar with vision.

In [13] Ye et al. present a framework for vision-based interactive components design. Both Kjeldsen et al. and Ye et al. propose to analyze camera input only in regions of interest around interactive "zones" of the interface, thus making the vision engine more economic in CPU usage. However Ye et al. go a step further, and instead of applying one vision algorithm per widget type, they propose to use a set of different detection techniques to obtain visual events called Visual Interface Cues (VIC). Each VIC detector called also *selector* examines a small part of the camera image for visual events that can be of different nature, like color, texture, motion or object geometry. Selectors are structured into hierarchies, and each selector can trigger one or more other selectors. Interaction events are detected based on sequences of selectors output. For instance, a touch sensitive button uses only motion detection at first. Once motion is detected, a VIC-based button switches to color-based image segmentation and to segmentation-based gesture recognition.

While the VIC paradigm presented in [13] is appealing for widget-oriented interface design, it lacks the analysis of VIC selectors suitability for interactive systems. Indeed, the VICs framework is presented from the perspective of an expert in computer vision. It does not consider setup and maintenance issues related to each selector such as threshold setting or lighting requirements. In section 4 we present a VIC-based implementation of basic interactive widgets that was conceived to respect the requirements identified in section 3.

Finally, the Papier-Mâché [6] toolkit features a framework for the creation of multimodal interactive applications, in particular using computer vision. The toolkit design is based on a detailed user study, and offers an abstract, event-based model to work with multiple modalities. The vision part allows object recognition and tracking to be used as application input. While the design seems to be well-thought from the HCI perspective, it has several architectural shortcomings; in particular typical problems linked to object recognition (such as the aspect variations due to lighting changes) are not dealt with, and several thresholds must be set manually, on-line, by the developer using the toolkit.

3. Service-oriented design

Vision-based user interfaces rely on vision processes that extract high-level, abstract information from streams of images. This information is what is relevant for the interaction task. Except for "proof of concept" demonstrations created by computer vision researchers, this information is to be used by interactive application developers who have little to no expertise in the computer vision. This implies that a vision system that aims to be used outside the laboratory must be designed from a developer-centric point of view. In addition, all interactive systems, including vision based once, must consider end-user requirements. In this section, we describe how user-centric and developer-centric requirements influence both the structure of a VB system and its application programming interface (API).

Our goal is to provide a set of computer vision services to developers of interactive systems. We aim at offering these services at different levels of abstraction. At the highest level, a service is an autonomous "black-box" application that, once launched, provides abstract information through some communication channel (a network socket, or a shared memory for examples). Developers should not need any knowledge of the vision processes. They simply connect to the black-box application and receive abstract events such as the position of the fingers in the case of a finger tracker service, or activation of striplets in the case of the SPOD service that we describe later. We facilitate the communication between the vision application and the developer's application by using a very simple communication protocol. This approach is developer-centric in the sense that we only expose and make easy to access what matters to the developer: the high-level events.

3.1. Non-functional requirements

Typical user-centric criteria against which an interactive system is evaluated include overall latency, reliability and autonomy [7]. Here, we shortly describe these criteria and how they constrain the architecture of vision systems.

Latency. Interactive systems generally require a constraint on latency. For user input systems latency is measured as the time between the user action and the notification of the application. Typically, when using a vision-based finger tracker or a mouse to drag projected objects on a surface, the latency must be under 50 ms to optimize usability [8]. On the other hand, for a system that monitors the number of persons present in a room, a latency of 1 second may be acceptable. In consequence, a general architecture for vision services must be able to respect strict latency limits in communicating information to the application.

Autonomy. Vision-based systems usually require operator intervention for initial setup and maintenance. While acceptable in a laboratory setting, such intervention is poorly suited for deployment. Our "black-box" approach reduces the possibility of operator maintenance by reducing communication between the vision service and the developer's application to a minimum.

Ideally, vision system developers should devise automatic maintenance solutions without offering a maintenance API. However, we acknowledge that designing fully autonomous vision system is a difficult goal that will not be reached in the short run for many services. When operator intervention is mandatory, one must take into account that the calibration task will be performed by a non-expert. Therefore, the maintenance procedure must be designed appropriately to minimize disrupting the end-user from her/his primary tasks.

Reliability. In most cases, information needs to be reliably conveyed between the input system and the application. For instance, an application that receives events when a person enters or exits an augmented room cannot afford to "miss" an event because its state would become incoherent. Losing some events from a tracking system, on the other hand, would not necessarily break the interaction, but would deteriorate the end-user experience. Therefore, our architecture must include reliable transport for interactive events.

3.2 Functional requirements

We propose a number of developer-centric requirements for our system in order to successfully address the target audience. These requirements include abstraction, isolation, and contract. Together they form the notion of a service.

Abstraction. To HCI developers, "abstracting the input [is] the most time consuming and challenging piece of application development" [6]. Since we assume the user of our vision system (the application developer) has no vision expertise, vision-specific information should not be made visible.

Even though coupling processing results with a confidence factor might be richer, this information is, in most cases, of no interest to developers. For instance, optical mice use normalized cross-correlation to determine the direction and speed of the movement. As long as the correlation coefficient is above a threshold they send positional data to the computer, otherwise they remain silent. The correlation coefficient is not sent to the system. The services internal data, such as the confidence factor, should not be considered as first-class output of services and should not be made visible by default. However, in cases where developers need to access this information, it should be available by explicit request from the developer.

End-user input should be abstracted to interaction events that are the same for a given class of input. For example, a finger tracker should produce "motion" events compatible with mouse "motion" events so that developers can, as a first step, quickly substitute vision-based input to existing event processing code. In other words, the abstracted data from the vision service should be in a form that is easily connectable to existing graphical user interface code. Consequently, the API of a VB input system for interaction should *(a)* render the vision aspects invisible, and *(b)* generalize the input used for a given task.

Isolation. Input subsystems like a VB-UI may need to be used by multiple applications, possibly running on different machines – for performance or geographic reasons. For instance, a video capture service might be used concurrently by a surveillance system located on a remote server, by a video-conference application, and by a motion capture service, both co-located to the camera. Since "a particular piece of input can be used for many different types of output" [6], information generated by a vision system must be shareable, and both remotely and locally accessible. Moreover, any input system for interaction should be easily extensible (for instance using output adapters, aggregators, or supervisor patterns) and embeddable in other services or in applications. Designing a VB input system as a federation of "black box" components therefore appears to be adequate.

Contract. Developers of toolkits or input devices usually establish a contract with the UI developer, in terms of HCI-centric, non-functional criteria. For instance, for positional or tracking input (mice, trackpads, laser trackers, etc.), the relevant criteria are latency, precision, robustness, and autonomy.

- *latency* is implicitly under a usability threshold for standard tracking input devices (the textbook value [8] is around 50 ms);
- *precision* is also implicitly defined for a mouse/trackpad (under 1 display pixel, scaled with respect to the device gain)
- *robustness* is generally "absolute". In other words the device or system either works or doesn't. For instance an optical mouse "just works" as long as used on an adequate surface, and stops emitting positional events as soon as the contact stops.
- *autonomy*, or the lack of setup and maintenance, is also tacit for traditional devices.

In the case of VB input systems, these criteria are difficult to meet. Therefore, it is the role of the vision system designer to explicitly state in contract form the limitations with respect to these criteria. For instance, the system should notify the client application about robustness-related failures rather than provide incorrect or distorted information along with a (low) confidence factor. This corresponds to a binary quality of service evaluation and requires the system to perform introspection.

3.3. A pragmatic approach

Our approach is to generally isolate as much as possible the VB input system from the application that uses it, and minimize the communications between them. We propose to encapsulate the relevant services into independent, "black-box" processes that use only serialized communications.

To allow for low latency while preserving reliability, we propose to use traditional socket-based (TCP and UDP) communications. We use the TCP link to guarantee the connection between services or with the application, and to allow for reliable, high-latency communications, and the UDP link for low-latency communications. This constitutes the base of the "BIP/1.0" protocol we use for communications between services (draft specification at http://www-prima.inrialpes.fr/prima/pub/Publications/index.php).

System autonomy cannot be enforced at the architectural level. Nevertheless, not providing support for synchronous communications (thus making it difficult to implement transactions between the application and a service) limits the possibility of implementing setup mechanisms where automatic setup can be devised.

Finally, since the *contract* offered by a VB input system cannot be expressed simply as a part of an API, we propose to document it explicitly. This means that the developer of the system must evaluate it against the criteria presented above in order to circumscribe conditions of use for the vision system.

4. Basic services for vision-based UI design

To illustrate our service-oriented approach with an implementation, we choose to study vision systems applied to traditional interfaces. Typically, we investigate the use of graphical WIMP-like interfaces projected on surfaces of mundane objects. We assume that end-users are free to interact with projected images without wearing markers or actuating any hardware.

We break down our VB input system into three services: *(1)* the widget service, which the client will interoperate with; *(2)* the image capture service, which abstracts the camera, allowing it to be used by other systems eventually; and *(3)* the calibration service, which establishes the geometrical mapping between the camera view and the display.

4.1. Simple Pattern Occlusion Detectors

In [3] we presented an appearance-based implementation of touch sensitive projected buttons which we called "Sensitive widgets". The presence of an object over a button on the interaction surface is detected by observing the change of perceived luminance over the button center area with respect to a reference area. By defining the reference area around the central-one, the button is made robust to complete occlusion, and sensitive to appearance changes made by elongated objects. Very simple image processing allows this system to maintain several dozens of sensitive widgets at camera frame rate (PAL-size images at 25 Hz) on a typical desktop computer. Moreover, this approach is robust to lighting changes, and thus suitable for configurations in which the interface is projected frontally from a video projector.

Implementation and evaluation of several interface prototypes based on sensitive widgets demonstrated that, from the end-user's perspective, robustness to partial occlusions is also necessary. Indeed, a user pointing at a far-part of the interface would likely hover her/his arm over projected buttons, thus triggering partially occluded widgets. A partial, unsatisfactory solution was obtained by deactivating partly occluded widgets based on the input from widgets placed further away from the user. The idea of combining inputs from several sensitive widgets led us to re-think the touch detection approach.

We now choose to assemble atomic occlusion detectors, which are to be placed within and around widgets, in a way that allows the system to distinguish some simple occlusion patterns. The geometry of the detectors (called "striplets") is simplified to a rectangular strip. These detectors are assembled into a federation to provide the SPOD (Simple Pattern Occlusion Detectors) service. The SPOD service is internally divided into two separated layers: the image processing layer called the Striplet Engine (SE), in charge of image processing, and the Vision Events Interpretation Layer (VEIL), in charge of input abstraction. Both of these layers were designed to meet the requirements described in section 3.

Striplets are defined as sensitive patches on the interface. Their response is calculated as the integral of the perceived luminance multiplied by a gain function over the surface of the striplet. The gain function has to be chosen so that the integral equals zero when the luminance over the whole striplet is constant. In current implementation the gain function is a symmetric step function with positive value over the central part of the striplet and negative value at both ends of the striplet (Figure 1).

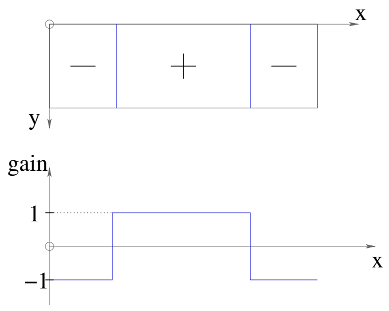

Figure 1. Striplet geometry and the gain function.

Striplets are designed to detect occlusion by elongated objects. Each striplet is 40 millimeters long and 10 millimeters wide on the projected interface. These dimensions are chosen to ensure a maximal response to occlusions made by finger-sized objects occluding the striplet's central area. Occlusions of any extremity of a striplet are intentionally ignored.

The camera coordinates of a striplet are calculated based on its position in the interface as given by the VEIL service and the camera-interface mapping. The event trigger threshold, on the other hand, is estimated individually for each striplet by the SE without any control from the client application. This threshold is automatically set to half of the maximal positive response of a striplet during interaction. The only assumption is that fingers contrast with the interface, which is true in most setups.

The VEIL is the "brain" of the SPOD service. The VEIL *(a)* translates widgets coordinates defined by the client application to a set of striplets coordinates, and *(b)* analyses the occlusion events generated by the SE and issues interaction events when appropriate.

Currently two types of interactive widgets are implemented: touch buttons and sliders. This allows a designer to build simple WIMP-like user interfaces. The button widget is composed of six striplets: two crossed in the center and four other surrounding the button center (Figure 2). Touch events are issued only if occlusion is detected by at least one of the center striplets and no more than one surrounding striplets. Sliders are simply obtained by assembling multiple partially overlapping buttons.

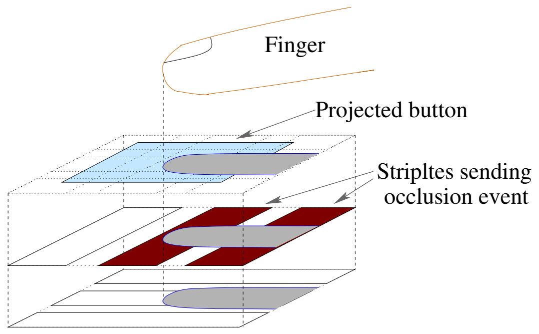

Figure 2. Button widget made of six striplets.

Since monocular vision systems cannot detect when a finger actually touches the interface, interaction events are generated only after detecting a short pause of the finger over a widget. The dwelling period, set to 250 ms, corresponds to the button-down event for mouse-based interaction. To move a slider, the end-user has to first "press" it and then drag it. This is coherent with existing WIMP interface behavior. In contrast, the dragging task requires from the end-user 2D movement coordination. If user's finger exits the SPOD slider area, dragging is stopped.

By assembling striplet detectors in more sophisticated ways, it is also possible to develop different types of interactive widgets, for instance crossing-based menus [1]. An extreme case would cover the whole interface surface with SPOD-button-like structures, thus making a SPOD-based finger tracker.

Service API. The SPOD service requires the client application to specify the position of each interactive widget in a normalized coordinate frame of the interface. Additionally, the SPOD service needs to know the mapping between the camera view and the interface, as well as a rough estimate of the number of pixels per unit length of the interface in the scene. Both the mapping and the scale, are provided by a calibration service (discussed below). The communication between SPOD service and calibration service is invisible to the client application. The SPOD service exclusively sends to the client application a stream of interaction events. All communication occurs via TCP/IP connections, using the BIP protocol described in section 3.3.

Inter-layer API. Both the SE and the VEIL are implemented as independent services, running in independent processes. Their communication also is asynchronous and event-driven, using BIP. For a given interface configuration, the VEIL sends to the SE coordinates of all striplets together with the interface-camera mapping matrix. The SE layer sends back to VEIL striplets state-change events that result from end-user interaction with the system.

Contract for the SPOD service. The initial SPOD service implementation can handle up to 300 striplets at camera frame rate (30 Hz) with images of 320x240 pixels size on a 2.8 GHz Pentium IV processor. In terms of widgets, this means the system can handle roughly 50 SPOD buttons simultaneously.

Because actuating buttons is not close coupled interaction, the latency is less of an issue. In fact, the VEIL makes the distinction between accidental occlusions and intentional actions based on a dwell time. On the other hand, the SE service is implemented to minimize latency.

Striplets only provide coarse resolution for finger positions. The resolution can be enhanced by averaging the position of several striplets detecting the same finger. Our implementation of a slider widget achieves a resolution of about 5 millimeters.

The SPOD service is made autonomous (i.e. exept for the UI-camera mapping and scale there are no parameters to set), at the expense of robustness to certain condition changes. In particular, the SE layer would fail to detect occlusion from a finger on a dark background, it would also fail if the image contrast decreases due to a change in camera setting.

While the SPOD service was designed to respect the developer-centric requirements, it does not fully meet user-centric requirements. In particular, the simplistic automatic threshold estimation results in occasional false positive touch detections. However, the SPOD service can be described within the Visual Interface Cues (VICs) framework [13], with local luminance changes as the only visual cue. Using a similar approach for spacio-temporal gesture recognition like in [13], we can hope to alleviate the need of setting an occlusion threshold. Instead, striplets responses would be fed to a neural network based VEIL.

4.2. Support Services

Calibration consists in the geometrical coupling of the camera view (what the vision system percieves) and the displayed interface (what the application developer controls). The calibration service clients need to access the mapping information to transform vision information (e.g. positions in camera coordinates) into application-relevant data (e.g. positions in interface coordinates). This is achieved by providing the associated projective transformation in matrix form. Because the calibration service needs information from both the camera and the application, two approaches are possible: *(a)* If it controls the graphical output, it can work without interaction with the application. This is the case in the PDS example [3], where the interactive surface itself is tracked by the service. *(b)* In the general case, it must negotiate with the client application the display of a calibration grid [7].

Image acquisition service creates an abstraction of the camera. It allows concurrent access to the camera by multiple services. In our case, both the calibration service and the SE require access to the image stream. Low latency video sharing is implemented using shared memory buffers.

5. Application

Using the widget implementation described above, we have implemented a simple calculator application. The calculator interface can be projected and manipulated directly with fingers on the top of a desk. This interface allows a user to perform basic calculations such as addition, subtraction, division and multiplication. Numbers can be either typed on the calculator keyboard or chosen from the history buffer containing results of previous operations. The history buffer can be browsed using a slider on the left side of the calculator.

An informal evaluation of the calculator application, made by volunteers from our laboratory, showed that SPOD-widgets are easy to use and allow fast interface prototyping. A video showing the application working is available in the demo section of `http://www-prima.inrialpes.fr/`

6. Conclusions

This paper presents a developer oriented design approach for vision-based interactive systems. Inspired by [5], we decompose the vision-based applications to isolated processes of vision components and functional core of the application. The implementation of the vision-components draws on the VICs framework presented by Ye et. al in [13].

We believe that extending these two design approaches by an HCI-centric requirements analysis allows to build vision systems that can be used for interactive systems designed by developers unfamiliar with vision. Following the guidelines of the developer-centric and end-user-centric requirements analysis we implemented vision-based interactive widgets: buttons and sliders. We illustrate the feasibility of our approach with an implementation of a simple calculator for projection-augmented surfaces.

Acknowledgments

This work has been partially funded by the European project FAME (IST-2000-28323), the FGnet working group (IST-2000-26434), and the RNTL/Proact ContAct project.

References

[1] G. Apitz and F. Guimbretière. Crossy: a crossing-based drawing application. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 3–12, 2004.

[2] F. Bérard. The magic table: Computer-vision based augmentation of a whiteboard for creative meetings. In *Proceedings of the ICCV Workshop on Projector-Camera Systems*. IEEE Computer Society Press, 2003.

[3] S. Borkowski, J. Letessier, and J. L. Crowley. Spatial control of interactive surfaces in an augmented environment. In R. Bastide, P. A. Palanque, and J. Roth, editors, *EHCI/DS-VIS*, volume 3425 of *Lecture Notes in Computer Science*, pages 228–244. Springer, 2004.

[4] J. L. Crowley, J. H. Piater, M. Vincze, and L. Paletta, editors. *Computer Vision Systems, Third International Conference, ICVS 2003, Graz, Austria, April 1-3, 2003, Proceedings*, volume 2626 of *Lecture Notes in Computer Science*. Springer, 2003.

[5] R. Kjeldsen, A. Levas, and C. S. Pinhanez. Dynamically reconfigurable vision-based user interfaces. In Crowley et al. [4], pages 323–332.

[6] S. R. Klemmer, J. Li, J. Lin, and J. A. Landay. Papier-mâché: toolkit support for tangible input. In *Proceedings of the 2004 conference on Human factors in computing systems (CHI'04)*, pages 399–406. ACM Press, 2004.

[7] J. Letessier and F. Bérard. Visual tracking of bare fingers for interactive surfaces. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 119–122. ACM Press, 2004.

[8] I. S. MacKenzie and C. Ware. Lag as a determinant of human performance in interactive systems. In *Proceedings of the conference on Human factors in computing systems*, pages 488–493. Addison-Wesley Longman Publishing Co., Inc., 1993.

[9] C. Pinhanez, R. Kjeldsen, T. Levas, G. Pingali, M. Podlaseck, and P. Chou. Ubiquitous interactive graphics. In *IBM Research Report RC22495 (W0205-143)*, May 2002.

[10] R. S. Rao, K. Conn, S. H. Jung, J. Katupitiya, T. Kientz, V. Kumar, J. Ostrowski, S. Patel, and C. J. Taylor. Human robot interaction: Application to smart wheelchairs. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA' 02)*, pages 3583–3589, May 2002.

[11] R. Raskar, J. van Baar, P. Beardsley, T. Willwacher, S. Rao, and C. Forlines. iLamps: geometrically aware and self-configuring projectors. *ACM Trans. Graph.*, 22(3):809–818, 2003.

[12] P. Wellner. The digitaldesk calculator: Tactile manipulation on a desk top display. In *ACM Symposium on User Interface Software and Technology*, pages 27–33, 1991.

[13] G. Ye, J. J. Corso, D. Burschka, and G. D. Hager. Vics: A modular vision-based hci framework. In Crowley et al. [4], pages 257–267.

[14] Z. Zhang, Y. Wu, Y. Shan, and S. Shafer. Visual panel: virtual mouse, keyboard and 3d controller with an ordinary piece of paper. In *Proceedings of the 2001 workshop on Percetive user interfaces*, pages 1–8. ACM Press, 2001.