Interacting with Projected Media on Deformable Surfaces

Fitriani
School of Computer Engineering
Nanyang Technological University, Singapore
fitr001@ntu.edu.sg

Wooi-Boon Goh
School of Computer Engineering
Nanyang Technological University, Singapore
aswbgoh@ntu.edu.sg

Abstract

This paper presents a novel human-computer interface for projector/camera-based applications that uses a deformable interaction surface. We discuss its design and implementation within the context of a radically different approach for controlling home appliances by pressing virtual buttons that are projected on soft deformable surfaces such as a sofa pillow. Effective real-time computer vision algorithms for implementing pointing and selection action detection for such an interface are discussed. Experimental results highlight the parameters and factors that have significant effect on the overall performance of such an interface.

1. Introduction

Vision-based human computer interaction (HCI) techniques [1,2,3] are often featured in pervasive computing applications as visual feedback provides very strong interaction cues, especially when the human user signals his intentions with actions involving limb movements or body posture changes. On another front, the cost and size of LCD projectors are continuing to fall, thus increasing the possibility of its ubiquitous deployment in the near future. This paper explores the implementation of a projector-camera based interface using deformable interaction surfaces.

A common man-machine interface found in a living room is the remote controller for home-entertainment appliances such as televisions or DVD players. For this reason, we decided to investigate the issues related to the design of the proposed interface within the context of such an application. The practicality of such a device replacing existing handheld remote controllers is neither the premise nor proposition of this paper. Instead, it focuses on the design considerations and real-time vision algorithms required to implement such a virtual remote controller (VRC), where a simplified button layout is projected on a pillow (see Fig. 3b). The required control functions associated with each button is activated by merely placing a finger over the desired image of the projected virtual button and pressing down on the pillow. The VRC requires a user to point to one of N buttons and make a deliberate depression of the button to indicate the selection of a desired operation. This translates to the problem of detecting pointing and selection (i.e. “click”) actions. Kjeldsen and Hartman [1] suggested that the design of control action must be intuitive, stable, comfortable, visible, and allows sufficient range of motion and multiplexing. The action should also be unique enough to prevent the “Midas Touch” problem, where everything the user does is interpreted as a selection action. A mere pointing action cannot be used as the control action to activate the virtual button since users may move their fingers around the buttons without the intention of activating any of them. Thus, a more deliberate click action is needed as a selection action. We posit that a deformable surface (e.g. pillow) is in fact a more appropriate interface medium than a rigid surface (e.g. table), since the depress-bounce-back effect approximates the action of pressing-releasing a button. Additionally, it is likely that soft deformable surfaces are present in a typical cozy living room environment, where ubiquitous HCI may occur.

2. Previous work

There has been a numerous research on vision-based finger-pointing [4,5] and hand gesture recognition [5,6]. Some of the sample applications include finger driven mice [7,8], finger driven drawing applications [9,10,11], and bare-hand television control [12]. This section surveys some of the approaches that have been used for detecting pointing and selection actions.

Aiming some extended body part, such as the nose, face or a finger at the desired location is the most intuitive pointing action. In the context of our application, the finger would be considered more appropriate than say using facial pointing [13], which would be better for hands-free computer screen-based pointing tasks. A fingertip detection algorithm in [5] first uses image differencing and then checks for filled and unfilled pixels to detect the fingertip. Unfortunately, such detection methods require sufficient contrast between fingertip and
the background. This is difficult to achieve under the low ambient lighting conditions.

The “click” action may be designed as a button touch event. Pinhanez et al. [4] defines a button touch event to occur when a fingertip is in the vicinity of a button, travels away from the user to a point within the button and then returns toward the user. Such restrictive algorithm can fail if the user moves his finger around before touching the button. Moreover, touching several buttons without retracting the user’s hand can result in confusing interpretations. A. Kale et al. [14] attempts to overcome this limitation by exploiting shadows to expedite the detection of a button touch event. The problem with using button touch is that touching a button may not equate selecting a desired function (e.g. the VRC). A more deliberate selection action such as a button press can help avoid the Midas touch problem. The FingerMouse system [5] detects a button click when a finger is on the same button for a one-second duration. However, time delay-based “click” events have limited usability in applications that require a fast response or rapid multiple clicks of the same button (e.g. reducing the TV’s volume quickly).

Hand gestures may also be used as a selection action [15, 16]. However, the task of hand segmentation in unconstrained environment and poor lighting conditions is difficult. Freeman et al. [12] developed an application to control the television set by using hand gesture to point and select the graphical remote controller image overlaid on the television screen. The user needs to hold up his hand, open it, move it right or left and close it to confirm the selection. This approach is neither intuitive nor comfortable for prolonged use. Moreover, the cognitive model for such a vision-based interface is too distance from the existing paradigm of a physical remote controller to be readily adopted. We posit that the approach proposed here is more intuitive than Freeman et al.’s approach [12].

Our proposed pointing and selection event detection strategies are designed with the environmental conditions we envisage the VRC to be deployed in. Firstly, we assumed large ambient light intensity changes in a living room situation where a user may turn down the room lights when watching TV. This prevents the use of fast skin tone detection techniques [17] for pointing detection as it requires favorable lighting conditions and good image contrast between object and background. Secondly, in most living room environment, there is a prevalence of soft surfaces, such as the sofa’s arm rest or a cushion pillow. We take advantage of this to implement robust click detection when the surface is intentionally deformed. Such click events are more intuitive and deliberate since the user can feel the pressing action rather than just a button touch or placing the finger over the button for certain period.

3. System overview and design

![Figure 1: The virtual remote controller system.](image)

Fig. 1 shows the overview of the VRC system. A projector is mounted on the ceiling and it projects the image of the virtual buttons on to the sofa’s arm rest or on a pillow on the user’s lap. A USB-based webcam that is placed behind the user’s shoulder captures the view of all the buttons. The webcam is placed above user’s head to prevent any occlusion by the user’s body. Both webcam and projector are interfaced to a standard personal computer (PC), which analyzes the webcam images in real-time. On detection of a button “click” event, the PC communicates the selected function to a USB-enabled PIC18F4450 microcontroller (µC), which has been interfaced to an off-the-shelf programmable remote controller (PRC). The appropriate button on the PRC is shorted by the µC’s output port so that the correct infrared modulation code is sent to the device based on the selected function (e.g. volume +).

![Figure 2: (a) The virtual button graphical design, each button n has its own gray base color. (b) Definition of button region (RBn) and button background region (RGn) for button n.](image)
Fig. 2a shows the layout of the virtual button design used. The graphics of each button \( n \) is defined as a rectangle and its internal color is a variable shade of gray with an intensity value of \( IB_n \), which is computed using a contrast optimization algorithm (see Sec. 3.1). Any symbol that represents the button’s functionality can be placed inside the button as long as it does not touch the button’s border. The symbol color is not constrained except for aesthetic and user visibility considerations. A black background is used to improve image contrast.

### 3.1. The activator spot

When the VRC system is first started up, a single white color spot of the same spatial dimensions as the virtual button is cyclically projected on to each of the position where the actual VRC buttons will be located (see Fig. 3a). This iconless white button is called an **activator spot** and it serves two functions. Firstly, the slow cyclical movement of the activator spot over the VRC active area allows a user to activate the VRC by placing his hand over the activator spot. If the activator spot’s internal intensity is deemed to have been changed (by the presence of the hand) for more than 2 seconds, the full VRC functionality is activated (see Fig. 3b). This strategy is essential to prevent projector burn-in that may result from the continuous display of the VRC buttons (much like the role of screen-savers) and it also serves to reduce the distraction of the VRC when not in use.

![Idle state](image1.png) ![Active state](image2.png)

Figure 3: (a) In idle state, an activator spot moves cyclically over the VRC’s button positions. (b) VRC goes into active state if a hand or finger is over the activator spot for > 2 secs.

A secondary purpose of the activator spot is to perform contrast optimization for each of VRC button. Due to the changes in the ambient light conditions and the automatic gain control (AGC) function of the webcam, it was observed that virtual button graphical layout with fixed button intensity results in images with poor contrast quality. Experimental results (see Sec. 4.1) showed that the contrast between the virtual button and its surrounding has a significant impact on performance of both the button detection and pointing action detection tasks. During the idle state, each time the activator spot relocates to a new button location, images of the projected activator spot are captured and analyzed. The internal intensity, \( IB_n \) ∈ [0,255] of the activator spot at the location of button \( n \) is progressively reduced from white to the gray until the contrast function given by

\[
Contrast = A(9RB_n) - A(9G_n) \quad (1)
\]

is maximized. The functions \( A(9RB_n) \) and \( A(9G_n) \) gives the average grayscale intensity observed in the activator spot region \( 9RB_n \) and its background region \( 9G_n \) respectively (see Fig. 2b). The \( IB_n \) value that maximizes the contrast in (1) at button \( n \)’s location is then assigned to the base color of the graphics of button \( n \) in Fig. 2a.

### 3.2. Button location detection

Fig. 4a shows an image of the virtual buttons captured by the webcam that is placed behind and above the user’s right shoulder. The reference edge \( R_n \) of the virtual button \( n \) is defined as the set of image points along the side of the rectangular button that is closest to the user (and camera) as highlighted by the solid black line in Fig. 4b. Based on the image coordinate reference axes shown, we define this edge as the bottom edge of the button. The line \( R_n \) of button \( n \) therefore starts with the bottom-left corner point \( (C_{BLn}) \) and terminates at the bottom-right corner point \( (C_{BRn}) \) of the connected closed contour. The importance of the edge \( R_n \) will be discussed in Section 3.3. With the webcam on the user’s right shoulder, another important parameter is the top-left corner \( (C_{TLn}) \), whose displacement is deemed informative in detecting button depression, as discussed in Sections 3.3 and 3.4. The top-right corner would be considered instead if the webcam be placed on the left shoulder of a left-handed user.

![Partial view of virtual buttons](image3.png)

Figure 4: (a) Partial view of virtual buttons captured when webcam is placed above user’s right shoulder. (b) The parameters to be extracted from each button \( n \) based on the image coordinate reference axes shown (see text for details).

Before the parameters shown in Fig. 4b can be obtained, the neutral positions of the \( N \) virtual buttons are located by finding all closed contours in the scene (using OpenCV’s `cvFindContours()` procedure in [18]). This initial stage of start-up calibration assumes no hand occlusions exist. Only contours with between 4 to 10 vertices are considered for further analysis. Since rectangular buttons are used, all selected contours are
reduced to 4 vertices by selecting top-left, top-right, bottom-left and bottom-right most vertices. Further reduction is done by analyzing the relative width ($w_n$) to height ($h_n$) relationships of each closed contour and the contrast between pixels in $RB_n$ and $RG_n$, till only $N$ most probable closed contours remain.

### 3.3. Pointing action detection

Techniques that detect the pointing action by tracking the fingertip using either image differencing or skin regions with template matching [1, 5] are not applicable due to the possibility of poor ambient lighting conditions. In dim light, the fingertip is not visible to the camera unless it falls within the brightly lit button interior. However, due to the contrast optimization between button interior and its background, the image quality of the finger in the lit area is often saturated making outline detection difficult (see Fig. 6a). Instead, a method which measures the distortion of a button’s edge is used to detect if the finger is currently located on top of one of the virtual buttons.

Scale normalization is done using a quickly computed vertical height of the button $n$ and is given by

$$h_n = y_{CTL_n} - y_{CBL_n}$$

where the 1D subtraction is between the $y$-axis components of the top-left ($CTL_n$) and bottom-left ($CBL_n$) corners of button $n$ as shown in Fig. 6a. The absolute height of the bump distortion is computed as

$$b_n = y_{max} - y_{min}$$

where $y_{max}$ and $y_{min}$ are the largest and smallest $y$-axis values on the distorted reference edge $R_n$.

Figure 6: (a) The parameters determining the normalized height of the bump-like distortion of reference edge $R_n$ when a finger is pointing at button $n$. Once pointing action is detected, the reference top-left corner point $C_{TL_n}(0)$ is noted so that (b) a corner displacement $d_n(f)$ can be computed when $C_{TL_n}(f)$ changes with each video frame $f$ during button depression.

A pointing event $P_n$ at button $n$ is deemed to be active (i.e. $P_n = 1$) if a finger currently straddles the reference edge $R_n$. As seen in Fig. 6a, this event can be detected by

$$P_n = \begin{cases} 1 & \text{if } B_n \geq (1/K_1) \\ 0 & \text{otherwise} \end{cases}$$

where $K_1$ is a sensitivity threshold for deciding the minimum height for the bump-like distortion before a pointing event is considered detected. Ideally, the value of $K_1$ should be made small (i.e. less sensitive) so that a large bump height is required before a pointing event is detected. This will reduce false detections due to creases on the pillow surface caused by a finger that is pressing in non-button regions. However, as can be observed in Fig. 6, the bump height $B_n$ reduces with increasing pressure being applied to the button surface. This is because the height of the finger relative to its surrounding surface has been correspondingly reduced with the increasing force of depression. Since the state of the pointing event $P_n$ should continue to be true even during button depression, this implies that the bump height threshold for detection must be made adaptive to the button depression pressure.

But how could one infer the depression pressure that is currently being applied? The extent of button depression at video frame $f$ can be observed by the amount the current top-left corner $C_{TL_n}(f)$ of the button has been displaced.
from its original position at $C_{TLn}(0)$, when no pressure was applied (see Fig. 7). Using the Euclidean displacement measure $d_n(f)$ shown in Fig. 6b, a modified pointing event detection scheme is employed after video frame $f=0$, when $P_n(0)$ first becomes 1

$$P_n(f) = \begin{cases} 1 & \text{if } [B_n \geq (1/K_1) - (d_n(f)/h_n)] \wedge [P_n(f-1) = 1] \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where the corner displacement $d_n$ is scale-normalized using $h_n$ in (3) and $\wedge$ is the logical AND operator. The proposed strategy for detecting pointing action events has some hysteresis-like property. A large distortion in the proposed strategy for detecting pointing action events has displacement condition at video frame to determine this fact. We can then formulate the extent of the displacement of the top-left corner of the button should allow us from its original pre-depression position should allow us to determine this fact. We can then formulate the extent of displacement condition at video frame $f$ (after the detection of the pointing event $P_n(0) = 1$ at $f=0$) as

$$E_n(f) = \begin{cases} 1 & \text{if } d_n(f) \geq h_n/K_2 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $h_n$ is the spatial scaling factor given in (3). The threshold $K_2$ is set based on the desired sensitivity of the system. A system that would activate the VRC buttons using only soft pressing would use a large value of $K_2$. If a more deliberate system is required, in which button would only activate when a significant amount of depression is needed, a smaller value of $K_2$ would then be used. In the system used for generating the experimental results in Section 4, a value $K_2 = 2$ was used. In this case, a depression of about 30-40 mm is required to generate a selection action event since a very soft cushion pillow was used as the deformable surface. Should a more rigid upholstery be employed as the interaction surface, a larger value of $K_2$ would be recommended or else the selection action will not be detectable due to the shallower depth change during button depression (see experiment in Fig. 11a).

Another condition to reduce the Midas touch problem is to detect that the button depression is being made in a monotonic fashion. This means a continuous downward pressure is applied until a selection event is detected. This monotonic depression condition is formulated as

$$M_n(f) = \begin{cases} 1 & \text{if } [\frac{\partial d_n(f)}{\partial f}] \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where a positive rate of change of the corner displacement $d_n(f)$ with consecutive video frames $f$ implies a continuous downward motion of the button’s surface. When this occurs, the value of $M_n(f) = 1$.

Another important consideration is that the conditions stipulated in (7) and (8) are consistently observed over a sufficient number of video frames ($F$) after the detection of the pointing event $P_n(0) = 1$ at start frame $f=0$. This consideration can be viewed as some form of temporal low pass filtering and it has some effects in reducing erroneous selection event detection and is further discussed in Section 4.3. Now combining all the stated conditions earlier and also including the fact that the selection event $S_n(f)$ can only be considered after the accompanying pointing event $P_n(0) = 1$ at $f=0$, we can summarize the formulation for computing a selection event on button $n$ as

$$S_n(f) = \begin{cases} 0 & \text{for } f < F \\ \prod_{i=0}^{F} (P_n(i) \wedge E_n(i) \wedge M_n(i)) & \text{for } f \geq F \end{cases} \tag{9}$$

$$S_n(f) = 0 \quad \text{for } f < F$$

$$S_n(f) = \prod_{i=0}^{F} (P_n(i) \wedge E_n(i) \wedge M_n(i)) \quad \text{for } f \geq F \tag{10}$$

Figure 7: The progressive movement of the button’s top left corner $C_{TLn}(f)$ away from its original position $C_{TLn}(0)$ with increasing pressure. (a) Pointing with no pressing, (b) start pressing, (c) mild pressing and (d) hard pressing.

**3.4. Selection action detection**

The next task after detecting an active pointing event is to decide when a proper button selection event has occurred. In order to determine a genuine selection event, several conditions must be simultaneously observed. Firstly, we need to detect that the button surface has been depressed to a sufficient depth before a button selection is positively registered. As highlighted in Fig. 7b, the extent of the displacement of the top-left corner of the button from its original pre-depression position should allow us to determine this fact. We can then formulate the extent of displacement condition at video frame $f$ (after the
The product function $\Pi$ will only yield a 1 if all the logical conjunctions comprising of the logical AND of the three conditions $P_n(i)$, $E_n(i)$ and $M_n(i)$ are all 1 for each frame $f$ from $0..F$. Otherwise a value of 0 results. As such, the selection event $S_n(f)$ returns 1 if the virtual button $n$ that is being pointed to is depressed in a deliberate manner and with sufficient depth. The associated TV function that is selected can be transmitted remotely to the TV once when $S_n(f)$ first becomes 1 from its initial 0 state. An auto-repeat feature can also be implemented if so desired. The same selected command can be repeated at a preset frequency as long as $S_n(i) = 1$ for $i > F$. In this case the modified formulation for a selection event detection is given by

$$S_n(f) = \prod_{i=0}^{f-1} (P_n(i) \land E_n(i) \land M_n(i)) \quad \text{for } f > F \quad (11)$$

However, this feature was not implemented in the system used in the experiments presented in Section 4.

### 3.5. Release action detection

Besides the detection of a selection action event, the button release event must also be detected. This will facilitate the implementation of quick successive activation of the same virtual button, such as when performing TV channel scanning. The release event detection can only be considered after the associated selection event $S_n(f)$ goes from 1 to 0, at which point the video frame numbering is reset to $f=0$. Like the selection event detection function, the release event function also requires the pointing event $P_n(f)$ to continue to remain active but now the complement of both the extent of displacement $E_n(f)$ and the monotonic depression $M_n(f)$ conditions must be 1. The release event formulation is given by

$$R_n(f) = 0 \quad \text{for } f < F \quad (12)$$

$$R_n(f) = \prod_{i=0}^{F-1} (P_n(i) \land \overline{E_n(i)} \land \overline{M_n(i)}) \quad \text{for } f \geq F \quad (13)$$

A positive release event can only be detected when a total of $F$ successive video frames after $S_n(f) = 0$ have lapsed and during this period, all three conditions stipulated in (13) must continue to remain true.

### 4. Experimental results

A series of experiments were done on an IBM Pentium M notebook with 768 MB of memory and running at 1.3 GHz. A Creative Live! Ultra webcam with 640 x 480 pixel resolutions, 25 fps and built-in AGC was used, together with a 3M X55i LCD projector with 1024×768 XGA resolution. The program was written and compiled with Microsoft Visual C++ 6.0.

In the studies presented, two types of error rates are used. The first error rate measures the number of undetectable or incorrectly detected action events (i.e. pointing or selection) per $J$ recorded events in the test video series. This is termed the UID error rate. For example, if a pointing action event $P_1$ occurs at frame $f$ in the test video, but no pointing action was detected, this counts as one UID error. If $P_2$ was detected instead of $P_1$, this also counts as one UID error. UID error rates are listed as a percentage since the total expected positive events $J$ in the test video is known a priori. The second error rate measures the number of falsely detected action event that has occurred when no action event exists in a segment of the test video. This is termed FD errors. For example, a user may press a non-button region and a selection event $S_1$ is detected. This counts as one FD error. Since FD errors have no upper bound, they are described using discrete counts.

#### 4.1. Contrast of virtual buttons

Fig. 8 shows that increasing contrast values given in (1) can reduce the UID error rates when attempting to successfully detect all $N$ virtual buttons closed contours (in 200 trials) and detecting all 200 pointing action events in a test video. This is why contrast optimization is implemented by adjusting each button’s base color intensity ($IB_n$) before action event detection. Fig. 8 shows that reliable performance is only obtained when the average contrast between the button’s interior and background exceeds a value of 120 (for 8-bit pixel intensity measures of $[0,255]$).

![Figure 8: Button and pointing action detection error rates at various buttons interior to background contrast values.](image)

#### 4.2. Pointing action detection threshold $K_1$

The threshold $K_1$ presented in (5) and (6) determines how sensitive the pointing action detection algorithm should be. The larger the value of $K_1$, the more easily a mild distortion of the reference edge $R_n$ will be interpreted as a positive pointing action event. So it is not surprising...
to observe in Fig. 9b that as $K_1$ gets larger, the number of FD also increases. In such situations, erroneous button pointing events are detected when a hand moves over the VRC or a finger presses just outside the button causing slight distortion of button’s borders.

On the other hand, when $K_1$ is set too small, the number of UID errors increases since it is now more difficult to detect a pointing action event unless a very significant bump height is registered on the $R_e$ edge. Only at $K_1 = 4$ is the UID error rates minimum. As $K_1$ increases beyond 4, the increased sensitivity sometimes causes the pointing action event for button $A$ to be detected when the finger is actually pointing at button $B$ (i.e. incorrect detection). It can be concluded, that for the experimental setup used, the optimal setting for $K_1$ is 4 and this was adopted in all subsequent experiments.

4.3. Number of consecutive video frames $F$

Some form of temporal filtering was introduced in Eqns. (9) to (13) when trying to detect a button selection or release action event. The argument for such a strategy is that at a typical frame rate of 25 fps, it would generally require several frames of consistent depression before the displacement threshold for button selection in (7) is reached. By aggregating consistent observations over $F$ frames, sudden changes in VRC imagery (e.g. a hand waving over the buttons) would not cause false selection action detection. This strategy does work as observed by the falling number of falsely detected events when $F$ increases (see Fig. 10b). However, $F$ cannot increase indefinitely as UID errors will begin to occur (see Fig. 10a). Buttons may have been quickly pressed and released before the selected $F$ number of consecutive frames have lapsed, therefore missing the selection action event. The response speed of the VRC system to repeated pressing of the same virtual button is also related to the value of $F$. For fast down-up-down pressing of a button, the value of $F$ should be kept small. The results for a 25 fps system seems to suggest that the optimal value of $F = 5$ gives the least combined UID and FD errors. However, when the webcam experiences low frame rates (e.g. 15 fps), a value $F = 5$ will result in increased UID errors as the downward movement of virtual button at nominal pressing speed will be completed in less than the expected 5 consecutive video frames. Therefore, in cases of low frame rate, the value of $F$ should be reduced accordingly.

4.4. Other factors influencing performance

Some form of temporal filtering was introduced in Eqns. (9) to (13) when trying to detect a button selection or release action event. The argument for such a strategy is that at a typical frame rate of 25 fps, it would generally require several frames of consistent depression before the displacement threshold for button selection in (7) is reached. By aggregating consistent observations over $F$ frames, sudden changes in VRC imagery (e.g. a hand waving over the buttons) would not cause false selection action detection. This strategy does work as observed by the falling number of falsely detected events when $F$ increases (see Fig. 10b). However, $F$ cannot increase indefinitely as UID errors will begin to occur (see Fig. 10a). Buttons may have been quickly pressed and released before the selected $F$ number of consecutive frames have lapsed, therefore missing the selection action event. The response speed of the VRC system to repeated pressing of the same virtual button is also related to the value of $F$. For fast down-up-down pressing of a button, the value of $F$ should be kept small. The results for a 25 fps system seems to suggest that the optimal value of $F = 5$ gives the least combined UID and FD errors. However, when the webcam experiences low frame rates (e.g. 15 fps), a value $F = 5$ will result in increased UID errors as the downward movement of virtual button at nominal pressing speed will be completed in less than the expected 5 consecutive video frames. Therefore, in cases of low frame rate, the value of $F$ should be reduced accordingly.

Figure 9: (a) Undetected or incorrectly detected (UID) error rates for different values of $K_1$ and (b) number of falsely detected errors for different values of $K_1$. The test video used for this experiment contains 200 pointing action events. User was asked to do random pointing on all buttons.

Figure 10: (a) UID error rates for different values of consecutive frames $F$ and (b) number of FD errors for different values of $F$. The test video used contains 200 selection action events. User was asked to do random selection on all buttons as well as multiple selections on the same button.

Figure 11: UID error rates for experiments with different (a) surfaces deformability, (b) color of pillow, (c) order of clicking of buttons and (d) ambient lighting conditions. The average UID error rate for all the experiments shown here is 7.6%.

Fig. 11a shows that the system performance is dependent on the softness of the surface used. A softer surface will allow greater movement of the virtual button and therefore allow a more positive detection of selection action events. Without changing the depth sensitivity threshold $K_2$ in (7), in can be readily observed that UID errors increase with the rigidity of the deformable surface. The color of the projection surface also influences the error rates. Fig. 11b suggests that darker color provide for better contrast, especially if the ambient environment is brightly lit. The VRC also performs better if different buttons are activated one after another instead of the same
button being pressed repeatedly (see Fig. 11c). It was observed that for many soft surfaces, repeated deformations at the same locality leaves residual deformation on the surface after the pressure is removed. Results of Fig. 11d suggest that the VRC performs better in a bright environment than in a dark one. The increase in UID error rate in a dark environment is mainly due to the characteristic of the AGC function of the webcam. The frame rate drops significantly in a dark scene because the webcam required more time for the CCD circuit in the camera to charge up so that the average image level can still be maintained by the built-in AGC function. At lower frame rates, UID errors tend to increase if the observation duration measured in the frame count $F$ in (10) is not reduced accordingly (see Sec. 4.3). This problem can be easily addressed by using an adaptive value of $F$ based on the current video frame rate of the webcam.

5. Conclusions

We have described a new approach for HCI using a projector-webcam setup and soft deformable interaction surfaces. Various vision algorithms to detect pointing and selection action events were presented. Experimental results show an error rate for undetected or incorrectly detected action events of less than 5% can be currently achieved with the use of appropriate soft bouncy pillow of darker color. The proof-of-concept example VRC application has been designed to work in both bright and dark environments. A simplified layout of only six buttons was used in our initial experiments. Layout with 12 buttons has also been successfully implemented. In any case, interfaces with minimalist look and feel do appear less daunting to most people. Extended device functionality can be designed using hierarchical levels of buttons if so desired. In reality, with VRC-type applications, we envisage a combined use of both physical and virtual remote controllers. The original physical device is used for complex tasks like channel tuning and the VRC provides the commonly used functionalities like those in Fig. 2a. These same interfaces can be used to control lighting, shut window blinds and other home automation functionalities if so desired.

References