Recognizing Gestures on Projected Button Widgets with an RGB-D Camera Using a CNN

Patrick Chiu
FX Palo Alto Laboratory
Palo Alto, CA 94304, USA
chiu@fxpal.com

Chelhwon Kim
FX Palo Alto Laboratory
Palo Alto, CA 94304, USA
kim@fxpal.com

Hideto Oda
FX Palo Alto Laboratory
Palo Alto, CA 94304, USA
oda@fxpal.com

Abstract

Projector-camera systems can turn any surface such as tabletops and walls into an interactive display. A basic problem is to recognize the gesture actions on the projected UI widgets. Previous approaches using finger template matching or occlusion patterns have issues with environmental lighting conditions, artifacts and noise in the video images of a projection, and inaccuracies of depth cameras. In this work, we propose a new recognizer that employs a deep neural net with an RGB-D camera; specifically, we use a CNN (Convolutional Neural Network) with optical flow computed from the color and depth channels. We evaluated our method on a new dataset of RGB-D videos of 12 users interacting with buttons projected on a tabletop surface.

Author Keywords

Interactive surfaces; depth cameras; gesture recognition; convolutional neural network

CCS Concepts

• **Human-centered computing~Gestural input**

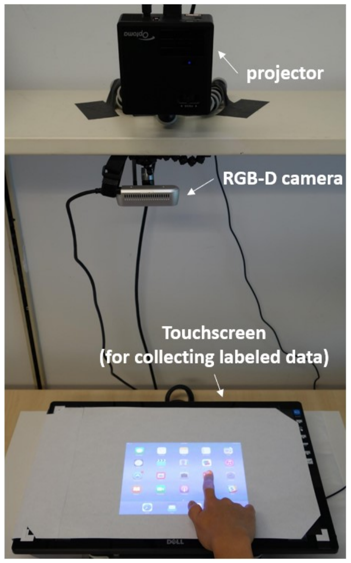


Figure 1: Hardware setup: Projector (Optoma ML500) and RGB-D camera (Intel D435) mounted on a shelf and pointing down on a tabletop surface to create an interactive display. For collecting labeled data, a touchscreen covered with white paper is used.

Introduction

Projector-camera systems can turn any surface such as tabletops and walls into an interactive display (e.g. [6], [14]). By projecting UI widgets onto the surfaces, users can interact with familiar graphical user interface elements such as buttons. For recognizing finger gesture actions on the widgets, computer vision methods can be applied, and RGB-D cameras with color and depth channels can also be employed to provide data with 3D information. A prototype of a projector-camera setup for an interactive tabletop display is shown in Figure 1.

A basic problem is to recognize the gesture actions with the UI widgets. There are several challenging issues with projector-camera systems and the environmental conditions. One issue is the lighting in the environment: brightness and reflections can impair video quality and make events difficult to recognize. Since the camera is pointed at a projection image, there can be artifacts like rolling bands or blocks that show up in the video frames, and this can cause unrecognizable or phantom events. With a standard camera (no depth info), all the video frames may need to be heavily processed, which uses up computing cycles. With a depth camera, there are inaccuracies, noise and artifacts (see Figure 2 bottom image), which can cause recognition errors.

In this work, we address these challenges using a deep neural net approach. Deep Learning is a state-of-the-art method that has achieved excellent results in a variety of AI domains including computer vision problems (e.g. [LeCun et al., 2015]). We apply a standard CNN (Convolutional Neural Network) with dense optical flow images computed from the color and depth video channels. Our method aggregates the frame regions around each button widget into events using a voting scheme, and the events are aggregated into gestures based on the UI layout of the widgets. Moreover, our processing pipeline also uses the depth info to filter out frames without activity near the display surface to reduce computation cycles.

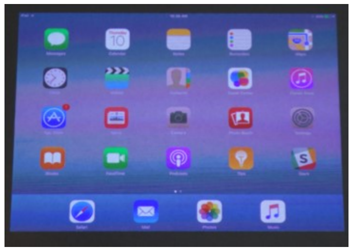
We evaluated our method using the latest version of the Intel RealSense RGB-D camera (D435) [4]. For collecting labeled data, we built a projector-camera setup with a special touchscreen surface to log the interaction events. We collected a set of gesture data with 12 users. The users interacted with buttons on a tablet style UI and a toolbar UI, and the gesture actions comprise 3 classes {Press, Swipe, Other}. The best optical flow component of our proposed method achieved 3.5% gesture error.

Our contributions include:

- A new method for recognizing gestures on button widgets for projector-camera systems based on deep neural networks
- A new dataset of RGB-D video frames of labeled gesture actions with button widgets on a surface

Related Work

In previous research systems, various computer vision and image processing techniques have been developed to detect gesture actions with button widgets for projector-camera systems. One approach is to model the finger (e.g. [3], [15]) or the arm [6], which typically involves some form of template matching. Another approach is to use occlusion patterns caused by the finger (e.g. [1], [12]).


(a)

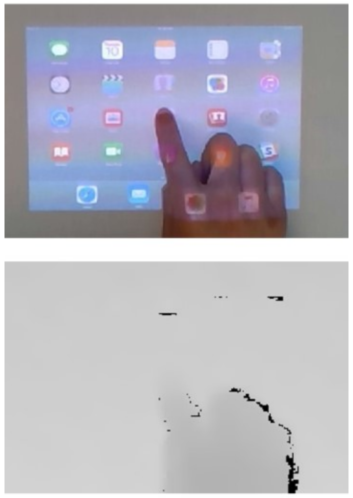

(b)

Figure 2: Projected UI of iPad style home screen image. (a) View as seen by user (from a photo). (b) Video frames from RGB-D camera, color and depth. These images have been cropped to save space.

For applying optical flow to action recognition, the method of [11] is used with normal video cameras but not with depth cameras and not with interaction on UI widget objects.

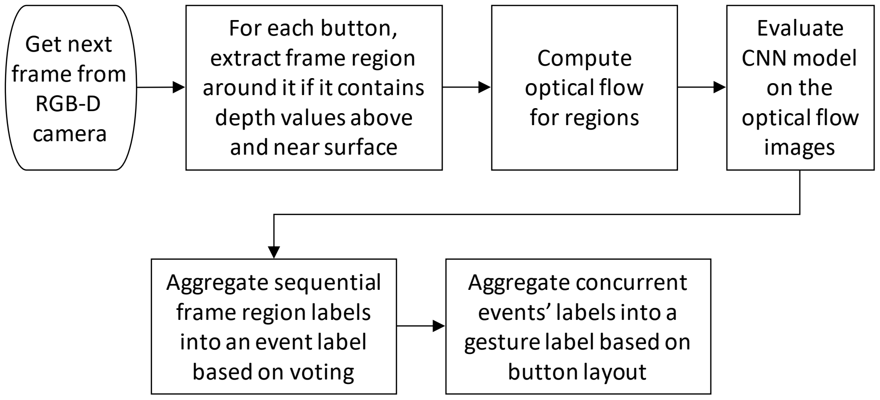
For RGB-D gesture action datasets, the MSR Daily Activity3D [13] is an example captured with a Kinect device (10 users, 16 activities, 2 reps). The BigHand2.2M [16] is a very large dataset (2.2M frames) of hand poses captured with the Intel RealSense device (SR300). These datasets do not have gesture actions with GUI widgets on a surface. We also use the new RealSense depth camera (D435), which produces less noise and artifacts (e.g. black areas in Figure 2b) than the previous version (SR300).

Gesture Recognizer Pipeline

The hardware setup with a projector and RGB-D camera is shown in Figure 1, and sample frames are shown in Figure 2. A diagram describing the video frame processing pipeline is shown in Figure 3.

The first part of the proposed pipeline uses the depth information to check whether something is near the surface on top of a region R centered at each button widget. To sense some of the surrounding action, we set R to be a square of size about four finger widths (~80 mm). The z-values of a small subsample of pixels $\{P_i\}$ in R can be checked to see if they are above the surface and within some threshold of the surface's z-value. If not, no further processing is required, and this saves computation cycles.

Next, the dense optical flow is computed over each frame region R for the color and depth channels. One motivation for using optical flow is that it is robust against different background scenes, which in our case means different user interface designs and appearances. The optical flow approach has been shown to work successfully for action recognition in videos [11]. To compute the optical flow, we use the Farneback algorithm [2] in the OpenCV library [9]. The



Get next frame from RGB-D camera
For each button, extract frame region around it if it contains depth values above and near surface
Compute optical flow for regions
Evaluate CNN model on the optical flow images
Aggregate sequential frame region labels into an event label based on voting
Aggregate concurrent events' labels into a gesture label based on button layout

Figure 3: Proposed pipeline for recognizing gestures on button widget.

optical flow processing produces an x-component image and a y-component image for each channel: {c0, c1} for color and {d0, d1} for depth.

We classify these optical flow images using a CNN model as gesture actions on the buttons with labels {Press, Swipe, Other}. For the CNN, we employ a standard architecture with two alternating convolution and max-pooling layers, followed by a dense layer and a softmax layer, using the Microsoft Cognitive Toolkit (CNTK) [8], which is suitable for integration with interactive applications.

A contiguous sequence of frame regions with activity over them accumulate in a buffer to form events. Each event buffer is classified and given a label by taking a vote of its frame regions' classification labels. Each optical flow component is voted on separately.

Finally, concurrent button region events are aggregated into a gesture event associated with a target button. Typically, a single button is labeled as either Press or Swipe, and the other buttons are labeled as Other. These other events are caused by the fingers and hand intersecting the various buttons in the UI layout (e.g. Figure 2b). If the events are all labeled correctly, there is no problem. However, if more than one event is labeled as Press or Swipe, heuristics based on the layout, such as using the label of the top-left region (for a right-handed user) can be used to determine the correct target button.

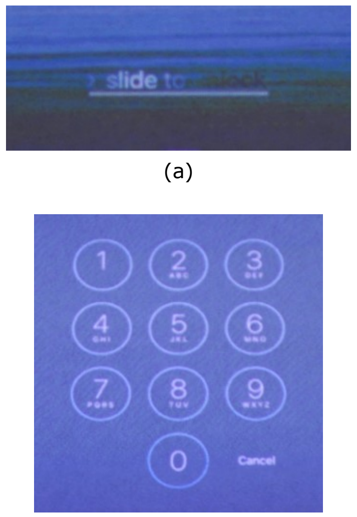


Figure 4: Projected iPad style widgets: (a) "slide to unlock" highlighted with underline, (b) "enter passcode" keypad. View as seen by user (from photos).

Collecting Labeled Data

For training and testing the network, we collected labeled data using a special setup with a projector-camera system, and a touchscreen covered with white paper on which the user interface is projected. See Figure 1. The RGB-D camera (Intel D435) is mounted 50 cm from the surface. The touchscreen (Dell S2240T) can sense touch events through the paper, and each touch event's timestamp and position are logged. The timestamped frames corresponding to the touch events are labeled according to the name of the pre-scripted tasks, and the regions around the widgets intersecting the positions are extracted. From the Intel D435 camera, we obtained frame rates that vary around 35-45 fps for both color and depth channels with the frames synchronized in time and spatially aligned.

We collected data from 12 participants. There are two parts totaling six tasks, with four gesture actions per task. The first part shows a tablet style UI, in which screenshots of iPad were projected on the surface. For the first screen ("slide to unlock"), the user was asked to make a swipe gesture (4 reps). For the second screen ("enter passcode"), the user was given a random 4-digit number and asked to enter it using tap gestures. For the third screen (home), the user was given a printout with 4 randomly highlighted icons and was asked to tap on them. See Figure 2 and Figure 4. The background images are scaled so that the projected buttons have the same size as on an iPad (15 x 15 mm). For the "slide to unlock" image, we added an underline to highlight the widget text to make it more visible (on the iPad the text is animated).

The second part shows a toolbar style UI, which we designed. Each button has a stripe on the bottom edge, and these buttons allow two types of interaction: swiping along the stripe or pressing on the button. The button size is 36 x 20 mm. In the toolbar, two buttons (far left and far right) are enabled and the middle three

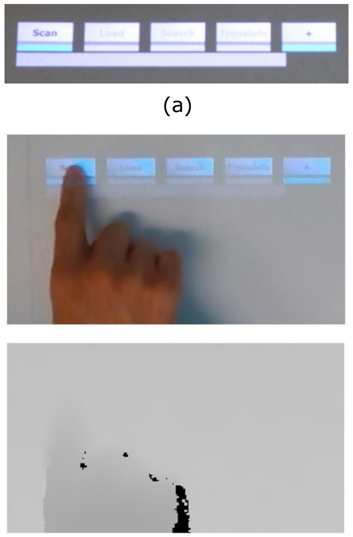


Figure 5: Projected UI of a custom toolbar with buttons. (a) View as seen by user (from a photo). (b) Video frames from RGB-D camera, color and depth. These images have been cropped to save space

buttons are disabled and grayed out. See Figure 5. The first task (2 reps) is for the user to swipe on the two active buttons, and the second task (2 reps) to press on the same two buttons. The third task (2 reps) is to use the palm to cover and press down over the same two buttons. Using the palm is a way to get a common type of bad events; this is similar to the "palm rejection" issue of tabletop touchscreens and pen tablets.

The total number of labeled events (gesture actions) for the 12 users is: 12*[(4+4+4) + (4+4+4)] = 288.

In addition, based on the proposed pipeline above, we detect other events when the depth value over the center of a button is within some threshold (~16mm) of the surface's z-value. This can occur when there are multiple rows of buttons as in the iPad home screen or the passcode keyboard (Figure 2 and Figure 5). When the user presses on a target button, some of the nearby buttons may be occluded. The number of these detected events is 280. The total number of labeled and detected events is 496.

A total of 5564 video frame image regions around the candidate buttons were extracted for each channel (color, depth) from the 496 events. For the swipe gesture, the touchscreen setup provided UP and DOWN timestamps that bound a time interval containing the frames of interest. For the press or tap gesture which usually registers as a single time point, we extracted the 10 frames (approximately 250 ms) centered at this time point. Each frame is labeled as one of three gesture classes {Press, Swipe, Other}.

Evaluation

We performed 4-fold cross validation. The dataset of 5564 frame image regions is partitioned into 4 subsets by cycling through the 496 events. The frames from each subset is used for testing for each round and the rest of the frames for training the CNN. Each optical flow component {c0, c1, d1, d1} is evaluated separately. The frame error is defined as the percentage of incorrectly classified frame labels, and the frame error after voting (Frame-V) is similarly defined. The gesture error is defined as the percentage of incorrectly classified gesture tasks performed by the user. These results are shown in Figure 6 and Table 1 (top half).

On the best optical flow stream (c0: color, x-component), the frame error 9.4% is reduced by the voting scheme to 1.5% event error, which rose up to 3.5% for the gesture error.

We also performed 4-fold cross validation across users to test how well the model works for unseen users. See Table 1 (bottom half). The errors are higher; on the best stream (U-c0: color, x-component), the frame error is 11.9% and the gesture error is 6.6%.

Conclusion & Future Work

We presented a new method to recognize gesture actions with UI widgets for projector-camera systems based on CNN and optical flow. We collected a new dataset of these interactions with labeled frames that are synchronized and aligned.

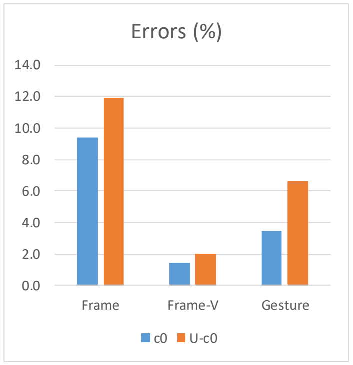


Figure 6: Results for color x-component of optical flow.

	Frame	Frame-V	Gesture
c0	9.4	1.5	3.5
c1	12.9	3.7	8.7
d0	18.6	6.6	16.0
d1	21.4	7.7	17.4
	Frame	Frame-V	Gesture
U-c0	11.9	2.0	6.6
U-c1	15.5	4.6	12.5
U-d0	22.1	10.0	20.1
U-d1	24.8	12.2	25.3

Table 1: Errors (%) for the different channel components.

For future work, we plan to supplement our basic method to by doing fusion on the optical flow streams, using a sequence of frames by extending the architecture to employ RNN networks, and incorporate spatial information from the frames.

References

1. Borkowski, S., Crowley, J.L., Letessier, J., Berard, F. User-centric design of a vision system for interactive applications. *Proc. ICVS '06*.
2. Farneback, G. Two-frame motion estimation based on polynomial expansion. *Proc. SCIA '03*, pp. 363-370.
3. Harrison, C., Benko, H., Wilson, A. OmniTouch: Wearable multitouch interaction everywhere. *Proc. UIST '11*, pp. 441-450.
4. Intel RealSense. https://software.intel.com/en-us/realsense/d400
5. Kinect. http://www.xbox.com/en-US/KINECT
6. Kjeldsen, R., C., Pingali, G., Hartman, J., Levas, T., Podlaseck, M. Interacting with steerable projected displays. *Proc. FGR '02*, pp. 402-407.
7. LeCun, Y., Bengio, Y., Hinton, G. Deep learning. *Nature*, vol. 521: 436–444 (2015).
8. Microsoft Cognitive Toolkit (CNTK). https://www.microsoft.com/en-us/cognitive-toolkit
9. OpenCV. http://opencv.willowgarage.com/wiki
10. Pinhanez, C., Kjeldsen, R., Tang, L., Levas, A., Podlaseck, M., Sukaviriya, N. and Pingali, G. Creating touch-screens anywhere with interactive projected displays. *Proc. ACM Multimedia '03 (Demo)*, pp. 460-461.
11. Simonyan, K., Zisserman, A. Two-stream convolutional networks for action recognition in videos. *Proc. NIPS '14*.
12. Tang, H., Chiu, P., Liu, Q. Gesture Viewport: Interacting with media content using finger gestures on any surface. *ICME '14 demo*.
13. Wang, J., Liu, Z., Wu, Y., Yuan, J. Mining actionlet ensemble for action recognition with depth cameras. *Proc. CVPR '12*, pp. 1290-1297.
14. Wellner, P. The DigitalDesk calculator: tangible manipulation on a desk top display. *Proc. UIST '91*, pp. 27-33.
15. Xiao, R., Harrison, C., Hudson, S. WorldKit: Rapid and easy creation of ad-hoc interactive applications on everyday surfaces. *Proc. CHI '13*, pp. 879–888.
16. Yuan, S., Ye, Q., Stenger, B., Jain, S., Kim, T.-K. BigHand2.2M Benchmark: Hand pose dataset and state of the art analysis. *Proc. CVPR '17*, pp. 2605-2613.