Fingertip Tracking for Tabletop Interaction Using a Depth Sensor

Anonymous

Anonymous

ABSTRACT

We report a new technique for using a depth sensor to track fingertips for tabletop interaction. Our technique is precise; the average position error (Euclidean distance) is 5.3px, about 10mm on the tabletop. We designed our system with naturalness in mind, making fewer assumptions about how users will move their hands on the tabletop. We believe our method is general and accurate enough to allow manipulative interaction using hands on a tabletop display.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*input devices and strategies, interaction styles*

General Terms

Human Factors

Keywords

Multi-modal interaction, natural human computer interaction, fingertip tracking, hand tracking, digital table interaction.

1. INTRODUCTION

Since its introduction, the Microsoft's Kinect has been used widely for capturing both body and hand gestures [1]. Many similar devices have subsequently come onto the market and have been used for capturing gestures. For instance, Harrison et al. [4] use a short-range PrimeSense [2] depth camera for their wearable multi-touch interaction. In comparison to the touch-sensitive devices, such as Microsoft's Surface and the iPad, depth sensors allow tracking of hands above the surface, and hence enable in-air gestures.

Our long term goal is to build an intelligent multi-modal interface for natural interaction that can serve as a testbed for enabling the formulation of a more principled system design framework for multi-modal HCI. One focus of the research is on gestural input for a large tabletop display. The user should be able to use gesture as an input effortlessly for both direct manipulation and communication to the system.

The first step towards gesture understanding is accurate hand tracking and hand model estimation. In this paper, we present our fingertip tracking technique using the Kinect sensor.

2. RELATED WORK

Hand tracking is essentially the frame-by-frame estimation of the parameters of the hand model. The complexity of the hand model is application dependent. For a given application, a very coarse and simple model may be sufficient. The simplest model treats the hand as a blob and only the 2D/3D location of the blob is tracked. For example, Sharma et al. [9] use 2D positional and time differential parameters to track the hands as blobs, which is sufficient for them to distinguish whether the hands are doing a point, a contour, or a circle gesture. The PrimeSense NITE middleware for OpenNI's natural interaction framework also tracks the hand as a point. It requires the user to do a "focus" gesture ("click" or "wave") to gain control in order to start hand tracking [8]. The gestures it supports are "click", "wave", "swipe left", "swipe right", "raise hand candidate" and "hand candidate moved".

However, to make a more generalizable system for natural-like interaction, a more sophisticated model is required. One step forward is adding fingertip locations in the hand model. Representative work in this area includes Oka et al. [6], which uses normalized correlation with a template of a properly sized circle to search for fingertips. They use a thermal camera to capture the hands. Larson et al. [5] also rely on a thermal camera, and use thinning operation to obtain a skeletization of the fingers, and then iteratively apply a hit-and-miss transform to find the endpoints of each finger. However, neither of them report the accuracy of their fingertip tracking.

Harrison et al. [4] use a depth sensor for fingertip tracking. They first compute the depth derivative of a scene, and look for vertical slices of cylinder-like objects. Then they group proximate slices into continuous paths and assume that the leftmost point in the path is the fingertip for a right-handed users. In their finger click evaluation, they report that their system has an average of 11.7mm offset from the targets. However, their method only works when the fingers approach the surface horizontally, not at an angle. This is a major assumption which may not be true for natural interaction. Our method does not make this assumption.

Tracking fingertips is usually sufficient for manipulating objects on a 2D surface [7]. A richer set of gestures, however, requires a more sophisticated model. For instance, Wang et al. [10] uses a 26 DOF 3D skeletal model in their real-time hand-tracking system. However, their system does not directly track the exact positions of the fingertips, and hence is not for touch-based interaction.

One requirement of our system is the ability for users to manipulate virtual 2D objects, for example, selecting, moving, and clicking on virtual objects, as for example, the urls in a web browser display. A simplified 3D skeletal hand model is needed for manipulative gestures because we want to know exactly where the fingertips are.

3. HAND TRACKING ON A TABLETOP DISPLAY

In this section, we describe our hand and fingertip tracking technique for a tabletop display.

3.1 System Setup

The custom tabletop structure includes four 1280×1024 pixel projectors (Dell 5100MP) that provide a 2560×2048 pixel resolution display. The display is projected onto a flat white surface digitizer (GTCO Calcomp DrawingBoard V), which uses a stylus as an input device. The digitizer is tilted 10 degrees down in front, and is placed at 41in (1.04m) above the floor, following FAA's design standard to accommodate the 5^{th} through 95^{th} percentiles of population. The projected displays were mechanically aligned to produce a single seamless large display area. The graphics card used is AMD Radeon^{TM}HD 6870 and the operating system used is Ubuntu 11.10.

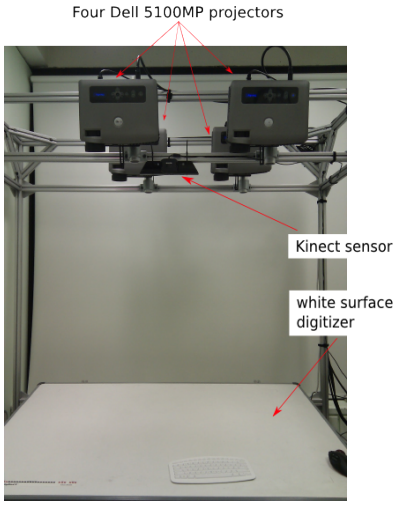


Figure 1: System setup.

One Kinect motion sensor by Microsoft is placed above the center of the tabletop at the same level of the projectors. Figure 1 shows the setup. The Kinect sensor has a RGB camera, a depth sensor and a multi-array microphone. We use the depth sensor to capture hand motion because it is less sensitive to the lighting condition. This is particularly useful for our projection system, as the Dell 5100MP projector uses a spinning color wheel to modulate the image, which produces a visible artifact on the screen with colors separating out in distinct red, green, and blue (see Figure 2). At any given instant in time, the image on the screen is either red, green, or blue, and the technology relies upon people's eyes not being able to detect rapid changes from one to the other. However, when seen through a RGB camera, the effect is very obvious, and this interferes with hand segmentation in RGB images. With a depth camera, we do not have this problem.

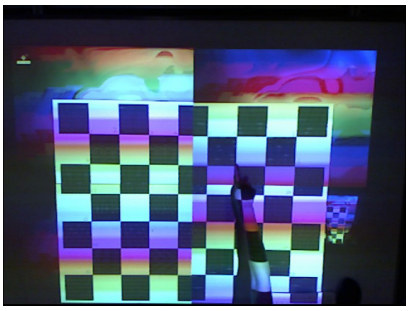


Figure 2: A RGB image with visible artifacts due to color-wheel projectors.

The Kinect sensor outputs video at a frame rate of 30Hz. The depth sensing video stream has a resolution of 640×480 pixels with 11-bit depth value. The depth sensor has a range limit of about 0.5m - 5m with a resolution of 1.5mm at 0.5m and 50mm at 5m. The tabletop surface is about 1.2m away from the Kinect sensor which allows us to have a relatively good depth resolution. We use the open source OpenNI framework [1], and its Kinect driver [2] to get both the depth and RGB data streams.

3.2 Kinect Calibration

In order to develop an interactive interface, we need to map a point in the depth image to a point on the display. We do this by projecting a checkerboard image on the tabletop display, and placing some wooden blocks at the corners of the checkerboard image to create the depth differences, allowing us to capture both positi0n and depth (see Figure 3). We manually labeled 16 pairs of corresponding points on the display and the depth image. Then we apply undistortion to the depth image and planar homography to find the mapping.

Planar homography is a projective mapping from one plane to another. In our case, we are mapping points on the plane of the depth image to the points on the plane of the display. To evaluate the result of the calibration, we obtain a new set of manually labeled corresponding points on the display and the depth image. We then transform the coordinates of the points on the depth image to the coordinates on the display using the calibration result, and find the Euclidean distance (error) between the transformed coordinates and the labeled coordinates of the points on the display. The average errors in the x-axis and y-axis are 4.3px and 8.9px respectively, which are 2.1mm, and 4mm in physical distance of the projected display. The average width of the human index fingertip is about 14mm, so the error is less than 30% of the width of the fingertip.

[1] `https://github.com/OpenNI/OpenNI`
[2] `https://github.com/avin2/SensorKinect`

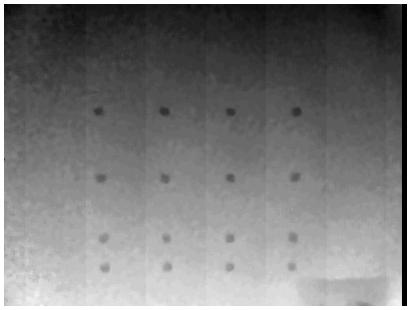


Figure 3: Kinect calibration. The darker squares are the wooden blocks. The intensity of the gray level image is inversely proportional to the distance from the Kinect sensor.

3.3 Hand Tracking

Our hand tracking process consists of feature detection and parameter estimation. The hand tracking pipeline consists the following steps:

1. Background subtraction
2. Upper limb and hand segmentation
3. Fingertips tracking

Many of the computer vision methods we use are based on the OpenCV [3] library and its Java interface JavaCV [4].

3.3.1 Background Subtraction

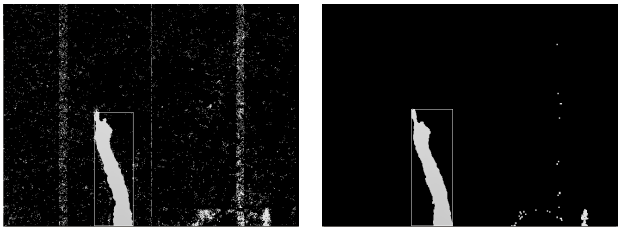
While the background - i.e., the tabletop - is static, there is still noise from the depth sensor. We use the averaging background method, which learns the average and average difference of each pixel in the depth image as the model of the background. The average values are based on the initial 30 frames with no hands in the scene; the average frame-to-frame absolute difference is 1.3mm in our case. For the subsequent frames, any value that is 6 times larger than this (i.e., at least 7.8mm above the surface) is considered foreground.

To clean up the background subtracted depth image, we use 1 iteration of morphological opening to clear out small pixel noise. Morphological opening is a combination of erosion and dilation. The kernel of erosion is a *local minimum* operator, while that of dilation is a *local maximum* operator. Figure 4 shows the difference after using morphological opening.

3.3.2 Upper Limb and Hand Segmentation

With the cleaned up background subtracted depth image, we find connected components by finding all contours with perimeters greater than a threshold value of 300mm. These components are considered to be the upper limbs. The threshold value is roughly the lower bound of the perimeter of a hand. We use this lower bound because the perimeter of the upper limb changes depending on the position of the hand relative to the table. The smallest perimeter occurs when the hand is at the edge of the table.

[3] `http://opencv.willowgarage.com/wiki/`
[4] `http://code.google.com/p/javacv/`



(a) Without using morphological opening.

(b) Using morphological opening to clear out small pixel noise.

Figure 4: Background subtraction.

Each upper limb is then approximated with a convex hull and a bounding box. The hand region is at either end of the bounding box depending on the position of the arm relative to the table.

3.3.3 Fingertip Tracking

We base our estimation of the hand model on geometric properties of the hand. We compute the convexity defects (shown by the white triangular areas in Figure 5) from the convex hull of the upper limb. These convexity defects offer a means of characterizing the hand shape [3]. As Figure 5 shows, an extended finger has one convexity defect on each side, and the two adjacent sides of the defects form a small acute angle (e.g., point A in Figure 5). We iterate through two adjacent convexity defects each time, for example, $\triangle B_i C_i D_i$ and $\triangle B_{i+1} C_{i+1} D_{i+1}$ in Figure 5(a). If the angle between the two sides, $C_i D_i$ and $B_{i+1} D_{i+1}$ is smaller than a threshold value ($45\,^\circ$), we mark the middle point of $C_i B_{i+1}$ as potential fingertips. The distance between the `depth_point`s (the point on the defect that is farthest from the edge of the hull [3], e.g., D_i, D_{i+1}) of the adjacent convexity defects also has to be greater than the finger width threshold value (14mm).

We further refine the fingertip position by searching in the direction of the finger for a sharp change in the gradient of the depth value (i.e., when the gradient is greater than 0.05).

3.3.4 Kalman Filtering

Like [6] and [4], we use a Kalman filter to further improve tracking accuracy. The dynamic state of the fingertip can be summarized by a 4-dimensional vector, x_k, of two position variables, x and y, and two velocities, v_x and v_y. The measurement is a 2-dimensional vector, z_k, of the measured x and y coordinates of the fingertip.

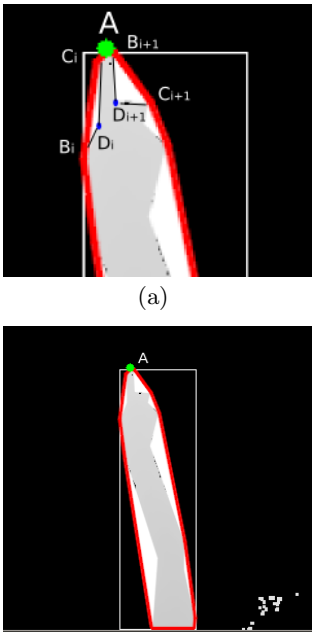
Assuming no external control, the a priori estimate x_k^- of the state is given by:

$$x_k^- = F x_{k-1} + w_k$$

F is the 4-by-4 *transfer matrix* characterizing the dynamics of the system with the following values:

$$x_k = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}_k, \quad F = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

assuming constant velocities, and that time is measured in



(a)

(b)

Figure 5: The white triangular areas are convexity defects and the red outline is the convex hull.

frames. w_k is the *process noise* associated with random events or forces that directly affect the actual state of the system. We assume that the components of w_k have Gaussian distribution $N(0, Q_k)$ for some 4-by-4 covariance matrix Q_k. We set the matrix Q with the following values:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

The larger variance values in the last 2 values in the diagonal indicate greater uncertainty in the velocities, as they may of course not be constant.

3.3.5 Evaluation

We evaluate the accuracy of our fingertip tracking method by comparing the detected fingertip position with the manually labeled position in a sequence of video frames. In our first evaluation, only one extended finger was used. Our method finds the fingertips with an average error (Euclidean distance) of 5.3px, about 10mm in physical distance on the projected display. The error rate also informs us the size of the virtual objects we should have in our applications, i.e., they need to be at least 10mm in radius, in order to increase the accuracy of the manipulative interaction. This result is comparable with the accuracy in [4], but our system has no restriction on the angle of the fingers with respect to the surface.

4. GESTURE INPUT FOR THE BROWSER

We are developing a browser-based game-like application that allows the use of hand gesture for interaction. With the recent development in HTML5, browser-based interaction has become richer and richer and a browser has become a routine application people use to do a variety of tasks. A browser-based game application easily enables online multiplayer interaction. So far, we have developed the infrastructure that can enable gesture input to a web application [5]. Our system uses WebSocket to establish the connection between the hand-tracking server and the browser application and transforms the coordinates from the virtual display coordinate to the browser coordinate.

5. DISCUSSION AND FUTURE WORK

We believe that, with the accuracy of our current fingertip tracking method, we can enable hand manipulation of virtual objects on a large tabletop display. We are going to develop demo browser-based applications to test this assumption.

Our fingertip tracking method does not have restrictions on the number of fingers we can track, but we still need to evaluate the accuracy for multi-finger tracking.

Eventually, we are going to develop a multi-modal system that allows both manipulative and communicative gestures, as well as speech input. The result of parameter estimation of the hand model will be used both as the information the system needs to make response to manipulative gestures, and as the input to a gesture recognizer.

6. REFERENCES

[1] OpenNI Arena. http://arena.openni.org/.

[2] PrimeSense Ltd. http://www.primesense.com.

[3] G. Bradski and A. Kaehler. *Learning OpenCV*. The Art of Computer Programming. O'Reilly, first edition, 2008.

[4] C. Harrison, H. Benko, and A. Wilson. Omnitouch: wearable multitouch interaction everywhere. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 441–450. ACM, 2011.

[5] E. Larson, G. Cohn, S. Gupta, X. Ren, B. Harrison, D. Fox, and S. Patel. Heatwave: thermal imaging for surface user interaction. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 2565–2574. ACM, 2011.

[6] K. Oka, Y. Sato, and H. Koike. Real-time fingertip tracking and gesture recognition. *IEEE Computer Graphics and Applications*, 22(6):64–71, 2002.

[7] 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*, 19:677–695, 1997.

[8] PrimeSence Inc. *PrimeSense^{TM} TM NIT Algorithms*, 1.5 edition, 2011.

[9] R. Sharma, J. Cai, S. Chakravarthy, I. Poddar, and Y. Sethi. Exploiting speech/gesture co-occurrence for improving continuous gesture recognition in weather narration. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 422–427. IEEE, 2000.

[10] R. Y. Wang and J. Popović. Real-time hand-tracking with a color glove. *ACM Transactions on Graphics*, 28(3), 2009.

[5]Code available at `https://github.com/xxxxx/xxxxxx` (anonymized for review)