PointPose: Finger Pose Estimation for Touch Input on Mobile Devices using a Depth Sensor

Sven Kratz, Patrick Chiu, Maribeth Back
FX Palo Alto Laboratory
3174 Porter Drive
Palo Alto, CA, 94304, USA
{lastname}@fxpal.com

ABSTRACT
The expressiveness of touch input can be increased by detecting additional finger pose information at the point of touch such as finger rotation and tilt. *PointPose* is a prototype that performs finger pose estimation at the location of touch using a short-range depth sensor viewing the touch screen of a mobile device. We present an algorithm that extracts finger rotation and tilt from a point cloud generated by a depth sensor oriented towards the device's touchscreen. The results of two user studies we conducted show that finger pose information can be extracted reliably using our proposed method. We show this for controlling rotation and tilt axes separately and also for combined input tasks using both axes. With the exception of the depth sensor, which is mounted directly on the mobile device, our approach does not require complex external tracking hardware, and, furthermore, external computation is unnecessary as the finger pose extraction algorithm can run directly on the mobile device. This makes PointPose ideal for prototyping and developing novel mobile user interfaces that use finger pose estimation.

Author Keywords
Mobile Device, Touch Input, Finger Pose, Depth Sensor, Point Cloud, Mobile Interaction

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION
Touch input is now the preferred input method on mobile devices such as smartphones or tablets. Touch is also gaining traction in the desktop segment and is also common for interaction with large table or wall-based displays. At present, the majority of touch displays can only recognize the touch location of a user input. Most capacitive touch screens can also report the contact area of a touch [2], but usually, no further information about individual touch inputs is available.

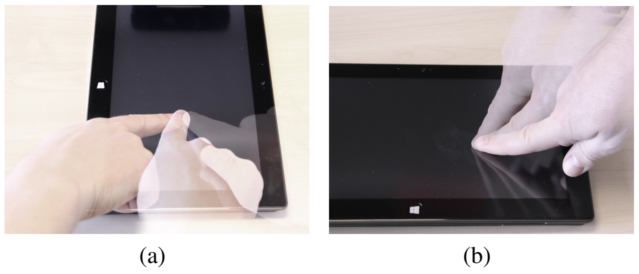


Figure 1. Using a short-range depth camera for sensing, our prototype is able to detect finger rotation (a) and tilt (b) on a mobile device equipped with a standard touchscreen.

It would, however, be beneficial to capture further properties of the user's touch, for instance the finger's rotation around the vertical axis (i.e., the axis orthogonal to the plane of the touch screen) as well as its tilt (Figure 1). Obtaining rotation and tilt information for a touch would allow for expressive localized input gestures as well as new types of on-screen widgets that make use of the additional local input degrees of freedom.

Estimating finger pose at a touch location increases the expressiveness of the user interface by providing additional local degrees of freedom of input at a given touch location. This, for instance, allows the user interface designer to remap established multi-touch gestures such as *pinch-to-zoom* [4] to other user interface functions or to free up screen space by allowing input (e.g., adjusting a slider value, scrolling a list, panning a map view, enlarging a picture) to be performed at a single touch location that usually need (multi-)touch gestures that require a significant amount of screen space. New graphical user interface widgets that make use of finger pose information, such as rolling context menus, hidden flaps or occlusion-aware widgets [20] have also been suggested.

In this work, we present a technique to estimate tilt and rotation parameters for touch input on mobile devices using a depth camera. We use a depth camera for sensing because the size of such devices has been decreasing over the past few years. Hardware manufacturers have already incorporated stereo camera pairs [8] and have suggested incorporating depth sensors [6] on mobile devices. Furthermore, the short-range sensing capabilities of depth cameras have improved dramatically, with finger-level sensing having recently become possible. We believe it is plausible that future mobile devices such as smart phones or tablets will be equipped with depth cameras. If these on-board depth sensors are placed at appropriate locations, or can be adjusted to a suitable orientation, it will be possible to augment the touch input capabilities of mobile devices with finger pose estimation using the technique we propose.

We contribute an algorithm for finger pose estimation that uses a point cloud obtained from a depth camera as input to obtain tilt and rotation values of a user's finger performing a touch. In contrast to previous work, our algorithm does not require a custom capacitive sensor [20], fingerprint scanners [11] or external overhead cameras [12]. Furthermore, our technique can cover the entire touchable area of a tablet, the only constraint being the field of vision of the depth camera. Moreover, existing devices can thus be extended with finger pose estimation capabilities by adding a depth camera. We demonstrate that our algorithm is efficient enough to run all point cloud processing tasks directly on the host device, making our prototype self-contained. We present the results of initial user study that provides insights into the tracking accuracy of our approach.

RELATED WORK

We have structured related work to cover previous contributions in the domains of *enhancing touch interactions on mobile devices*, *finger pose estimation* and *mobile user interfaces using depth sensors*.

Enhancing Touch Interaction on Mobile Devices

A significant amount of work has been conducted to enhance touch interaction on mobile devices. One of the problems of interaction with standard touch screens is that it is impossible to distinguish between dragging and tracking states according to the three-state model [3]. One solution to this problem is adding pressure sensing at the location of touch to enable a dragging state [24]. An alternative method is to use an external camera to sense hovering of the user's fingers over the touch screen [27].

The expressiveness of input at the touch location itself can also be improved. Boring et al. propose a method to use the contact size at the point of touch to add an additional degree of freedom to a touch input at a given location [2]. Further techniques that enhance touch input at the location of touch include circular motions [17], *Rubbing* [18], repeated tapping [18, 21] and very small rolling movements with the fingertip (*MicroRolls*) [22].

Finger Pose Estimation

Rogers et al. demonstrate the detection of finger tilt and rotation using a grid of capacitive sensors in combination with particle filters [20]. They apply their prototype to explore if touch precision can be improved by making use of finger pose information. With a similar goal, Holz et al. use an external camera to gather finger pose information to analyze how finger pose relates to perceived touch locations [11]. They

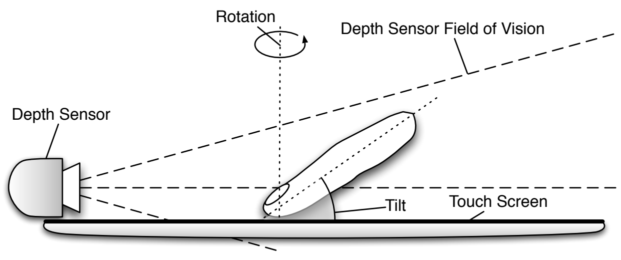


Figure 2. For finger pose detection, a depth camera is placed on or incorporated into the horizontal or vertical bezel of a touch screen device. This setup can be used to detect finger tilt and rotation.

use these insights in their implementation a highly accurate touch input device based on a commercial fingerprint scanner [12]. *ZTouch* [25] uses a multilayer array of illuminating lasers above a touch surface to infer the 3D position and pose of the user's fingers.

Pose estimation has also been explored with pen input. *GaussSense* [16] uses a grid of Hall sensors that is used to sense the 3D position and orientation of an input stylus. *SmartQuill* [28] uses acceleration sensors embedded within a digital pen to determine its tilt angle.

In contrast to previous systems we discussed for finger pose estimation, we argue that our system has the advantage of being lightweight: it does not need an external tracking system specially modified touchscreens—a suitable depth sensor clipped to any tablet-sized device will suffice. This makes our approach ideal for prototyping or research into touch input with finger pose estimation.

Mobile User Interfaces using Depth Sensors

A number of previous projects have made use of depth sensors for mobile user interfaces. *ShoeSense* [1] explores the use of a shoe-mounted depth camera to enable spatial gestures in an interaction volume covering the front upper body of the users. Some works [10, 15] explore the use of hand gestures in the volume immediately surrounding the mobile device. Harrison et al. [9] use a shoulder-worn depth camera to enable touch input on the user's arms and hands as well as objects in the environment.

IMPLEMENTATION OF THE POINTPOSE PROTOTYPE

Our proposed method estimates the finger pose (tilt, rotation; see Figure 1) of the user's finger at a touch location by processing point cloud data obtained from a depth sensor. In order for finger pose estimation using a depth camera to work, we must first consider the placement of the depth-imaging camera.

Figure 2 shows a possible placement of the depth-imaging camera in relation to the mobile device and the user's hands. The depth-imaging camera is incorporated into or attached to the mobile device's body with its field of view comprising the device's touch screen, and its orientation parallel to the plane of device's touch screen, in order to optimally capture the user's finger(s) while touching. This arrangement allows the detection of finger tilt relative to the touch screen as well as the finger's rotation around the axis perpendicular to the plane of the touch screen (rotation). Thus, two additional degrees of freedom of input are provided for each touch point.

Figure 3 shows an image of a hardware prototype of the system we developed. The current prototype consists of a short-range depth-imaging camera attached to Microsoft Surface Pro touchscreen device.

Finger Pose Estimation Algorithm

Our algorithm for finger pose estimation processes a point cloud[1] obtained from a depth sensor. It performs cylinder fitting on a subset of the point cloud, which corresponds to the user's finger touching the screen. We use the parameters of the fitted cylinder to estimate the finger's pose, in the rotation and roll axes with respect to the surface of the touch screen. Concretely, the algorithm performs the following calculation steps on a point cloud obtained from a depth sensor:

1. Filter the point cloud for the points that are located within the volume interesting for interaction. That is, all points located within the volume created by the bounds of the touch screen and the space up to 15 cm above the touch screen.
2. (Optional) Depending on the resolution of the depth camera, reduce the number of points in the point cloud to a manageable amount. One way to do this is to down sample the point cloud using a voxel grid.
3. Find the user's finger by searching for the point cloud point p_{closest} that is closest to the current touch location reported by the touch screen. Then, search for the k points in the point cloud nearest to p_{closest}. Similar to a flood-fill operation, this nearest-neighbor search will find the set of points Q_{finger}, that corresponds to the user's finger touching the touch screen.
4. Fit a cylindrical model to Q_{finger}, in order to estimate parameters for the finger's spatial orientation. This can, for instance, be accomplished using a RANSAC [7] approach. From RANSAC, we obtain the normalized axis C of the fitted cylinder.

To obtain tilt and rotation parameters from C, we look at the projected components C onto the plane of the touch screen. We define projection P of C onto the plane of the touch screen as:

$$P = C - (C \cdot Y)Y \tag{1}$$

Where $Y = (0, 1, 0)^{\intercal}$ is the vector direction of the y axis, which is also normal to the plane of the touchscreen.

We calculate the rotation angle ρ by calculating the angle of the projected component of P and the vector direction $X = (1, 0, 0)$ of the x axis:

$$p_X = P \cdot X \tag{2}$$

$$\rho = cos^{-1}(p_X) \tag{3}$$

[1]Most current depth cameras capture a 2D depth image with pixels containing depth values. To obtain a point cloud, the depth image pixels must be converted from image coordinates to points in world coordinates. Such conversion functions are usually found in the SDKs specific to the depth cameras used.

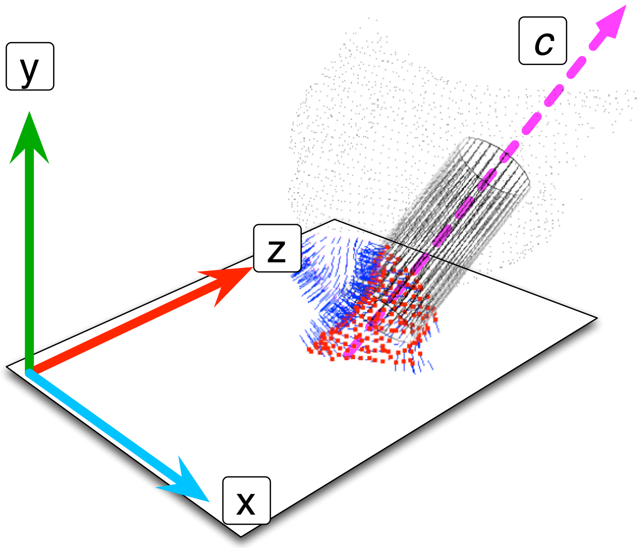


Figure 3. This figure shows point cloud of a user's touch captured by our depth sensor and the associated coordinate system. Our algorithm fits a cylinder on the subset of points that correspond to the user's finger executing the touch. The output of the algorithm is the orientation axis C of the cylinder, that is used for finger pose estimation.

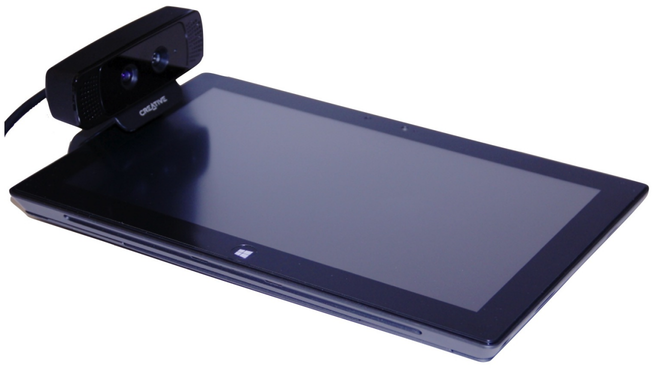


Figure 4. Our prototype hardware consists of a tablet with an attached short-range depth camera.

Similarly, for the tilt angle ϕ, we calculate the angle between the projection P of C on the plane of the touch screen and C:

$$p_C = P \cdot C \tag{4}$$

$$\phi = cos^{-1}(p_C) \tag{5}$$

Finger Pose Estimation Range

With the approach we have described previously, we can reliably estimate finger rotation over a range of 60 degrees and a finger tilt range of about 45 degrees. Due to the extreme finger tilt poses feeling uncomfortable for touch input, we constrain the range of finger tilt poses for our user studies to 25 degrees.

Software and Hardware Prototype

Our finger pose estimation algorithm is implemented in C++ making use of functions from the Point Cloud Library (PCL) [23] for point cloud processing. The finger pose estimation application is designed as a server that provides pose estimation information to client applications running as individual processes. This makes it easy to develop new applications using finger pose estimation or to include this functionality into existing application. The finger pose estimation server runs on Windows 8 on a Microsoft Surface Pro tablet computer. The prototype is thus fully self-contained and does not need external processing support. According to the Windows 8 Task Manager, the current finger pose tracker implementation uses about 35–40% of the available CPU[2] cycles.

We use a Creative Senz3D short-range depth camera for depth sensing. The Intel Perceptual Computing SDK [13] is used to access depth images provided by the camera and to generate world coordinates for depth pixels in order to create point clouds for use with the PCL. We use the tablet in portrait orientation the depth sensor clipped onto the top bezel. Figure 4 shows our current hardware setup.

EVALUATION

We conducted two user evaluations of the PointPose prototype, in order to examine how accurately finger pose input can be performed using our system. In the first study, we measure the accuracy for the rotation and tilt axes individually. In the second study we examine the accuracy of combined rotation and tilt input in a virtual joystick scenario.

User Study 1

In this study, we examine the accuracy of finger pose input in the rotation and tilt axes individually. The intent is to verify how accurate our finger tracking method is in each of these axes.

Subjects

A total of 8 subjects from an industrial research lab (4 female, one left-handed) participated in our study. The median age range of the participants was 32–36 years. All participants had experience in using touch-based mobile devices and all of them used such a device on a daily basis.

Apparatus and Task

We used the prototype hardware and finger pose estimation application described previously for this study. To smooth out tracking noise in the visualization, we used an adaptive low-pass filter. Our filter implementation, running an independent filter for each axis (rotation, tilt), was based on the *1€ Filter* [5] with parameters $\beta = 0.1$ and $f_{cmin} = 2.0$. However, the data we logged for further analysis remains unfiltered. We developed a user interface using Python with the PyGame library [19] to display and allow the user to interact with the study tasks. All tasks were displayed with the tablet in portrait orientation.

Figure 5 shows an screenshot of the task user interfaces we used in the study. The users were asked to perform target alignment tasks for rotation and tilt, respectively. For each task type, the user interface shows a controllable widget, sensitive to either rotation or tilt. Participants were asked

[2]The current Microsoft Surface Pro uses an Intel *Core i5* 3317U CPU at a clock speed of 1.7 to 2.6 GHz.

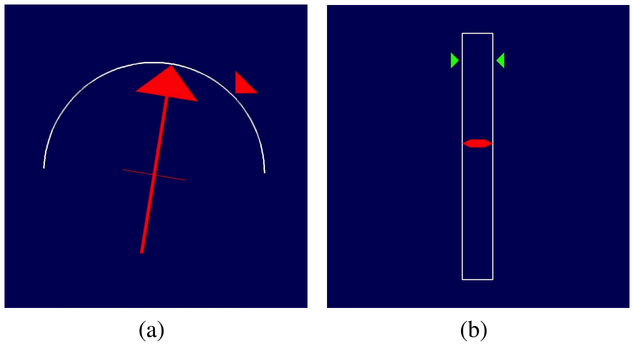


Figure 5. The widgets used for the tasks in first user study, showing a rotating arrow and target for rotation (a) and a vertically-moving wedge and target for tilt (b).

to make all touch-and-pose inputs in the lower half of the tablet's screen, in order to avoid occluding the widgets they were controlling. This widget had to be kept aligned with a target widget corresponding to a given angle for a particular trial and held at that position for 7.5 seconds.

For rotation, the controllable widget is an arrow that responds to finger rotation, that has to be aligned with a target triangle at a given target rotation angle (Figure 5(a)).

For tilt, the controllable widget is a bar that moves vertically, upwards when the finger is tilted upwards and downwards when the finger is tilted downwards. The task is to align the bar with a set of target triangles at a given target tilt angle (Figure 5(b)).

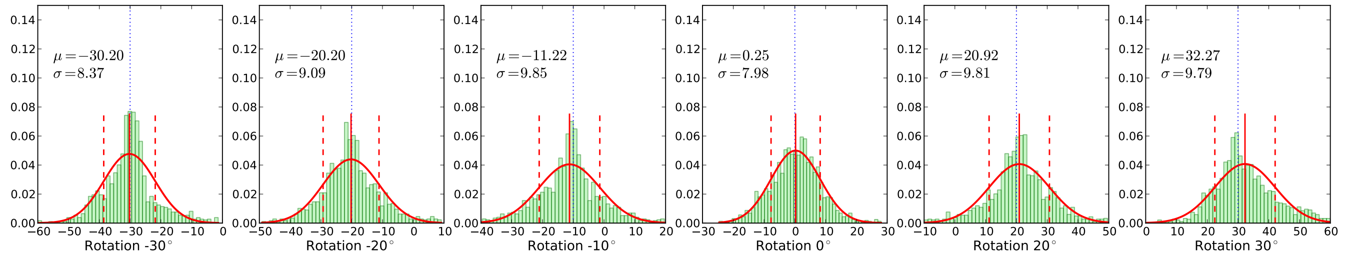
Conditions and Measurements

We defined a range of target angles for rotation and tilt, respectively. The tilt angles defined as targets were 50, 55, 60, 70 and 75 degrees. The rotation angles defined as targets were -30, -20, -10, 0, 10, 20 and 30 degrees, whereby -30 degrees corresponds to a finger rotated to the left and 0 degrees to the finger pointing towards the "12 o'clock" position, i.e., in a direction orthogonal to the tablet's top bezel. There were two repetitions for each target angle, thus we obtained data from a total of 80 and 112 input tasks for tilt and rotation, respectively. The type of input task and the ordering of the target angles was counterbalanced using a latin square design.

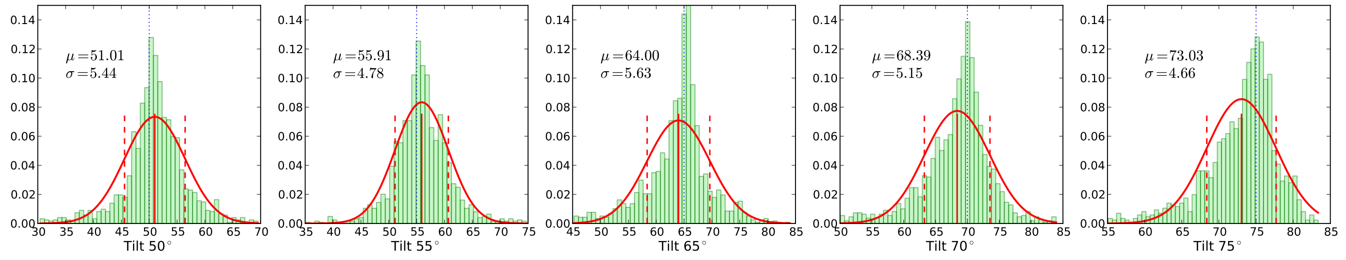
In order to determine the accuracy of finger pose input in rotation and tilt, we measured the position of the respective controllable widgets for 7.5 seconds once an alignment (within ± 5 degrees) with the target widget has been established. From the position information, we calculate the mean position and the standard deviation to gain insight into the accuracy of tilt and rotation input.

Results

From our study results, we calculated histograms of the controllable widget position during the 7.5 seconds of sampling. Figure 6 shows the histograms for tilt and rotation ordered by respective target angle. For rotation as well as tilt, the standard deviations do not vary substantially from each other, which indicates that all angles can be selected with more or less the same precision. The results for rotation show a higher



(a) Results for Rotation, by target angle



(b) Results for Tilt, by target angle

Figure 6. This figure shows histograms for the selection position for rotation (a) and tilt (b) by target angle. The dashed blue line represents the target angle, the solid red line the mean angle targeted by the participants and the dashed red line indicates one standard deviation of the measurements.

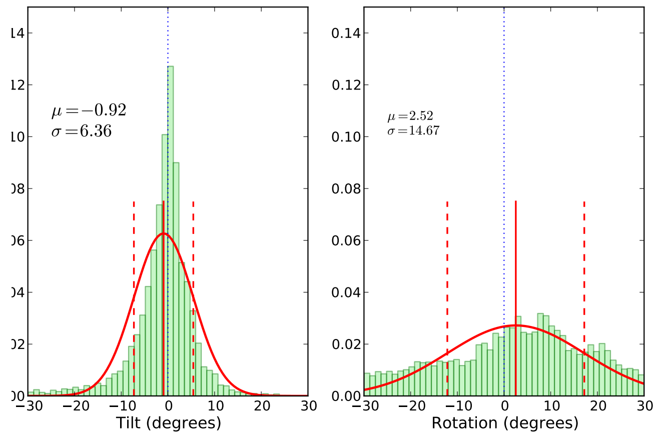


Figure 7. Global histograms for tilt and rotation input. The higher variance in rotation shows that the study subjects had more difficulty in properly targeting using that technique.

standard deviation than those for tilt, thus it appears that tilt is easier to control than rotation. However, the range of angles selectable by rotation (60°) are far greater than the one selectable via tilt (25°), so this may be a factor. Also, we cannot rule out the possibility that the finger movement for rotation is physically difficult for the participants. This is a question that needs to be addressed in future studies.

To obtain a somewhat broader view on the accuracy, we also calculated global histograms for tilt and rotation, which are shown in Figure 7. To generate these results, we offset all measured angles by the target angle for the respective conditions, such that the values are "normalized" with the target position at 0 degrees. For tilt, the mean was within 0.92 degrees of the target with a standard deviation of 6.36. For rotation, the mean was within 2.52 degrees of the target and the standard deviation was 14.67 degrees. Again, this shows that targeting using rotation was more difficult for the subjects than tilt.

The results of the first study indicate that finger pose estimation using a depth sensor is possible, but high variance, especially in the tilt axis shows that precision still needs to be improved. Tracking precision could be increased by using a depth sensor with a higher image resolution (when available) or using a more complex finger model (i.e., as in [20]).

User Study 2

From the results of the previous study, we see that finger rotation and tilt can be input reliably using our prototype, when controlling these axes individually. In this study, we examine the accuracy of finger pose input when rotation and tilt are combined.

Subjects

We recruited a total of 5 subjects from an industrial research lab (5 male, all right-handed). The median age range of the participants was 26–31 years. All participants had experience in using touch-based devices, all but one of the participants used their personal device on a daily basis.

Apparatus and Task

We used the same hardware as in the previous study. We also used the same adaptive low-pass filter [5] with parameters $\beta = 0.1$ and $f_{cmin} = 2.0$ to smooth the output for visualization. As in the previous study, we logged only unfiltered

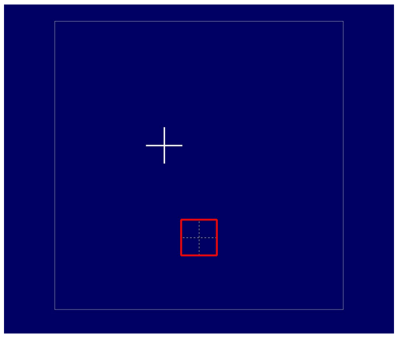


Figure 8. The widget displayed during the second user task. Using a virtual joystick metaphor, users had to move the control crosshair (white) towards the target region (red) and hold it at that location, using finger rotation and tilt.

input data. The user interface software for the tasks was developed using Python and the PyGame library. All tasks were displayed with the tablet in portrait orientation.

The task for this study was target alignment simultaneously in the rotation and tilt axes, using a virtual joystick metaphor. Figure 8 depicts one of the screens shown to the participants during the study. The task was to use finger rotation and tilt to maneuver a cursor towards a target crosshair and hold it at that location for 7.5 seconds.

Conditions and Measurements

Target locations were arranged on a 9×9 grid using -25,0 and 25 degrees for rotation and -10, 0 and 10 degrees for tilt. Subjects had to perform three repetitions per target, such that we captured a total of $9 \times 3 \times 5 = 81$ trials during the study. The ordering of the targets was counterbalanced using a latin square design.

We logged the position of the cursor controlled via finger tilt and rotation after it had reached alignment (within ± 5 degrees) with the target area. The logging duration was 7.5s each trial.

Results

Figure 9 shows an overview of the results we obtained for the second user study. The histogram of the input positions (Figure 9(a)) clearly shows that the users could align with the targets given during the trials, although the accuracy for the left row (finger rotated so that it is pointing towards the right of the touch point), in particular the $(-25, -10)$ target is noticeably lower than that for the other two rows. This may be due to the fact that this movement appeared more physiologically difficult for the (all right-handed) participants of this session. Alternatively, this may also indicate a weakness of our approach to finger pose estimation. A further investigation into the ergonomics of finger pose input, using a ground truth tracking system (e.g., VICON or OptiTrack) would thus be beneficial.

However, when looking at a global histogram (Figure 9(b)), offsetting the input values by the given targets to obtain a single target for all inputs, we can see that the users consistently managed to aim for their intended target. Separating the axis results for rotation (Figure 9(c)) and tilt (Figure 9(d)), we see that, as in the previous study, rotation control ($\mu = -0.03,\ \sigma = 10.03$) had a higher variance than tilt control ($\mu = -0.08, \sigma = 5.78$). However, precision of input was very high for both axes, with the averages being (globally) very close to their intended target locations.

We can see from our results that imprecision from tracking noise is still a problem using the hardware currently available to us. Nevertheless, we have demonstrated that finger pose estimation using a short-range depth sensor is feasible. While the tracking precision we have obtained could be further improved on, the precision is sufficient for our method to be used to prototype mobile user interfaces for touch-based mobile devices that make use of finger pose estimation.

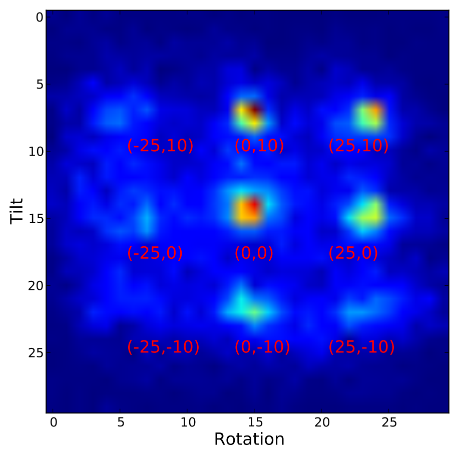
DISCUSSION AND FUTURE WORK

We presented a novel approach for finger pose estimation allowing the detection of finger rotation and tilt. In contrast to previous work, our approach uses a depth sensor to sense the orientation (rotation and tilt) of the user's finger. The algorithm we developed processes point cloud data obtained from the depth camera, and uses the touch location on the device's touch screen to determine the subset of the point cloud that corresponds to the user's finger touching the screen. Once this subset has been found, the algorithm determines the finger's orientation in space using RANSAC-based cylindrical fitting. The cylinder's orientation vector is projected onto the plane of the touch screen to determine rotation and tilt angles relative to the touch screen.

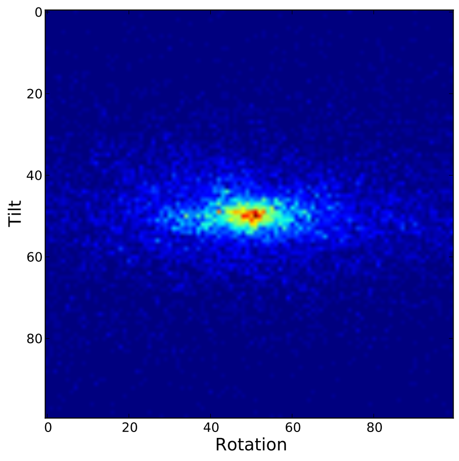
We conducted two user studies. In the first study, we analyzed the input precision of finger pose estimation when only one of the two axes (rotation or tilt) is controlled. In the second study, we examined the input precision when rotation and tilt are controlled simultaneously. Our results show that the users could consistently select and hold the target positions we had defined for them. However, we observed that tracking noise leads to a degree of variance in the input values, and thus effectively increases size of selectable targets using finger pose estimation for input. In both studies, the variance for rotation was about twice as high as that for tilt.

The advantage of the method we propose is that no specialized external tracking equipment is necessary and the tracking application can run directly on the tablet, with the exception, of course, of the short-range depth sensor, which is inexpensive, available commercially and can be attached to the mobile device to make the prototype portable. Furthermore, our algorithm is efficient enough to run directly on current mobile devices, so that there is no requirement to make use of external computational resources.

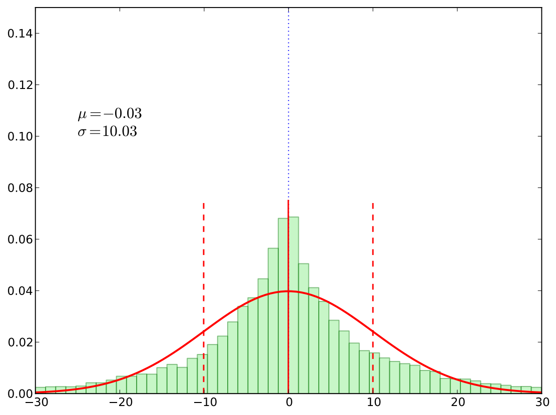
There is still room to improve our proposed technique. A more complex finger model could be fitted to the point cloud of the finger instead of the cylinder, which takes into account that fingers can bend on two joints. Furthermore, it would be relatively simple to extend our method to multi-touch, by selecting multiple sets from the point cloud that correspond to the fingers currently touching the device's surface. It is possible, however, that occlusion might become an issue in this case.



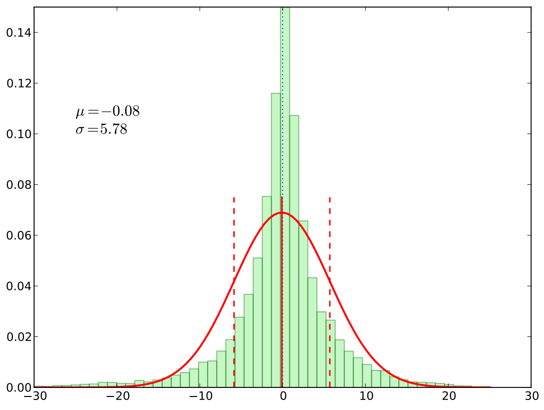
(a) Histogram 30 × 30 bins, 80 × 40 degrees.



(b) Combined histogram 100 × 100 bins, 60 × 60 degrees.



(c) Combined Rotation (degrees).



(d) Combined Tilt (degrees).

Figure 9. The results of the second user study. (a) shows a normalized histogram of the user inputs across all task. The peaks correspond to the target locations used in the task. (b) shows a histogram from the combined values of all trials. Separated results are shown for rotation (c) and tilt (d). As in the first study, rotation control showed a lower precision than tilt.

At present, the rate of technical evolution of depth sensing makes it is likely that depth cameras will soon be available on mobile devices. For future mobile devices, we envision a depth sensor mounted on a mobile device's bezel so that it can be swiveled either to face away from the mobile device to capture external scenes, or swiveled towards the touch screen to enhance input via finger pose estimation on the touch screen. The depth sensor placement we propose is not limited to applications for finger pose estimation. The availability of a point cloud containing details of the user's hands and fingers during interaction allows implementing further types of interactions, such around-device gestures [15] or touch-less interactions such as in-air password entry [14] (which mitigates the danger of smudge attacks [26]). In the future, we also intend to evaluate our method to obtain pose estimation for pen-based input, since most (non-reflective) pens are picked up in the depth image with a similar clarity similar to that of human fingers.

REFERENCES

1. Bailly, G., Müller, J., Rohs, M., Wigdor, D., and Kratz, S. Shoesense: a new perspective on gestural interaction and wearable applications. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, ACM (2012), 1239–1248.
2. Boring, S., Ledo, D., Chen, X., Marquardt, N., Tang, A., and Greenberg, S. The fat thumb: using the thumb's contact size for single-handed mobile interaction. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services*, ACM (2012), 39–48.
3. Buxton, W. A three-state model of graphical input. In *Proc. of the IFIP TC13 Third Intl. Conference on Human-Computer Interaction*, North-Holland Publishing Co. (1990), 449–456.
4. Buxton, W. Multi-Touch Systems that I have Known and Loved. `http://www.billbuxton.com/multitouchOverview.html`, June 2013.
5. Casiez, G., Roussel, N., and Vogel, D. 1 euro filter: a simple speed-based low-pass filter for noisy input in

interactive systems. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, ACM (2012), 2527–2530.

6. Enrico Guizzo, IEEE Robotics Blog. Hands-On With the Next Generation Kinect: PrimeSense Capri. `http://spectrum.ieee.org/automaton/robotics/robotics-hardware/handson-with-the-next-generation-kinect-primesense-capri`, May 2013.
7. Fischler, M. A., and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (1981), 381–395.
8. GSM Arena. HTC Evo 3D (Smartphone with Stereo Camera Pair). `http://www.gsmarena.com/htc_evo_3d-3901.php`, June 2013.
9. Harrison, C., Benko, H., and Wilson, A. D. Omnitouch: wearable multitouch interaction everywhere. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM (2011), 441–450.
10. Hasan, K., Ahlström, D., and Irani, P. Ad-binning: leveraging around device space for storing, browsing and retrieving mobile device content. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 899–908.
11. Holz, C., and Baudisch, P. The generalized perceived input point model and how to double touch accuracy by extracting fingerprints. In *Proceedings of the 28th international conference on Human factors in computing systems*, ACM (2010), 581–590.
12. Holz, C., and Baudisch, P. Understanding touch. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, ACM (2011), 2501–2510.
13. Intel. Intel Perceptual Computing SDK. `http://software.intel.com/en-us/vcsource/tools/perceptual-computing-sdk`, June 2013.
14. Ketabdar, H., Yüksel, K. A., Jahnbekam, A., Roshandel, M., and Skirpo, D. Magisign: User identification/authentication based on 3d around device magnetic signatures. In *UBICOMM 2010, The Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies* (2010), 31–34.
15. Kratz, S., Rohs, M., Guse, D., Müller, J., Bailly, G., and Nischt, M. Palmspace: continuous around-device gestures vs. multitouch for 3d rotation tasks on mobile devices. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, ACM (2012), 181–188.
16. Liang, R.-H., Cheng, K.-Y., Su, C.-H., Weng, C.-T., Chen, B.-Y., and Yang, D.-N. Gausssense: Attachable stylus sensing using magnetic sensor grid. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, ACM (2012), 319–326.
17. Moscovich, T., and Hughes, J. F. Navigating documents with the virtual scroll ring. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, ACM (2004), 57–60.
18. Olwal, A., Feiner, S., and Heyman, S. Rubbing and tapping for precise and rapid selection on touch-screen displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2008), 295–304.
19. pygame.org. Pygame game programming library. `http://www.pygame.org`, June 2013.
20. Rogers, S., Williamson, J., Stewart, C., and Murray-Smith, R. Anglepose: robust, precise capacitive touch tracking via 3d orientation estimation. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, ACM (2011), 2575–2584.
21. Roudaut, A., Huot, S., and Lecolinet, E. Taptap and magstick: improving one-handed target acquisition on small touch-screens. In *Proceedings of the working conference on Advanced visual interfaces*, ACM (2008), 146–153.
22. Roudaut, A., Lecolinet, E., and Guiard, Y. Microrolls: expanding touch-screen input vocabulary by distinguishing rolls vs. slides of the thumb. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2009), 927–936.
23. Rusu, R. B., and Cousins, S. 3d is here: Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE (2011), 1–4.
24. Stewart, C., Rohs, M., Kratz, S., and Essl, G. Characteristics of pressure-based input for mobile devices. In *Proceedings of the 28th international conference on Human factors in computing systems*, ACM (2010), 801–810.
25. Takeoka, Y., Miyaki, T., and Rekimoto, J. Z-touch: an infrastructure for 3d gesture interaction in the proximity of tabletop surfaces. In *ACM International Conference on Interactive Tabletops and Surfaces*, ACM (2010), 91–94.
26. von Zezschwitz, E., Koslow, A., De Luca, A., and Hussmann, H. Making graphic-based authentication secure against smudge attacks. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, ACM (2013), 277–286.
27. Wigdor, D., Forlines, C., Baudisch, P., Barnwell, J., and Shen, C. Lucid touch: a see-through mobile device. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, ACM (2007), 269–278.
28. Williams, L. SmartQuill. *British Telecom Labs* (1997).