Human Computer Interface Using Kinect Sensor

Anonymous ICCV submission

Paper ID ****< replace **** here and in header with paperID>

Abstract

Kinect was released by Microsoft in November 2010 at an the affordable price, with real time 3D color-depth reconstruction and human pose estimation. The sensor has been widely accepted by the computer vision community and used for 3D reconstruction, human pose tracking and human computer interface (HCI). In this paper, we focus on expanding the default Kinect HCI capabilities by developing a 3D hand tracking based HCI as well as a hands-free head tracking based HCI with gesture recognition. The proposed implementation works with multiple basic functionalities, such as, single/double click, swipe up/down and back according to different actions, respectively. Our experiments show that our method works in real time under real life noisy operational environments. Areas of interest for this type of HCI system include remote control of commercial TV/computer displays as well as for real-time remote control of robotics, such as online surgery, and HCI for handicapped ones.

1. Introduction

Human-Computer Interfaces (HCI) using video imagery is central to many applications, such as surgery inspection, robotic control, helping disabled individuals communicate, as well as for game control. The key preprocessing aspect of such a system is human detection by background subtraction as well as full or partial human pose estimation for gesture recognition. In recent years, various engineers and companies have developed HCI systems with several well-known examples, namely the Apple multi-touch gesture [1] and IBM linguistics control system **Error! Reference source not found.**.

Previous HCI approaches have relied on both passive and active illumination sensors as input for the preprocessing tasks of human detection pose estimation. Passive color and IR video imagery sensors have been used as pointer input devices by tracking body pose as well as eye gaze for computer control [2][2][4][5]. While systems for human pose detection typically are quite inexpensive, they tend to be sensitive to contrast and lighting conditions, leading to poor segmentation of humans subjects from the background, and limited to only recognizing certain body poses. Significant improvements were realized using calibrated multi-camera systems that performed stereo processing for 3D depth reconstruction. Passive stereo color-camera systems have been developed that create dense 3D maps, which in turn could be used for human detection and pose estimation. However, 3D depth processing with such systems is computationally expensive, leading to unacceptable interface lag. Sparse 3D depth computation helps to reduce the lag, however, there is loss of information that leads to a degradation of human detection and pose recognition.

Compared to body-pose tracking HCI systems, gaze-based HCI systems are much more expensive and typically require inconvenient experimental set-ups for gaze tracking. In addition, gaze HCI systems have several significant drawbacks with one of the most significant being lacks of accuracy due to the fovea limits and drift, informally known as the Midas Touch problem[6].

An alternative to multiple passive sensors is the use of active sensors, with illumination used to determine range depth. Examples of such systems are VICON IR sensors [7], as well as 3D laser radar (lidar) sensors, which have been used for human foreground detection and pose recognition. One advantage of an active sensor over a multi-camera passive sensor is a significant increase in range accuracy as well as a reduction in computational time for reduced interface lag. One large disadvantage is that they are typically much more expensive than passive camera systems, making them prohibitive for individual purchase.

Recently, Microsoft introduced the Kinect system, HCI system which combines both passive color video imagery as well as active illumination for real time computation of dense 3D depth imagery. Unlike previous 3D color-depth commercial products, the sensor is quite inexpensive and is marketed to a large population of computer gamers, leading to wide adoption of the platform. The wide adoption led to the development of open-source libraries to access the raw 3D color-depth data as well as the built-in human detection and pose estimation functionality,

making the system a good candidate for usage in an enhanced HCI interface.

In this paper, we focus on expanding the default Kinect HCI capabilities by developing 3D hand and head tracking algorithms with action recognition. The proposed implementation recognizes a typical set of actions for interfacing with a modern operating system (OS) graphical user interface (GUI), namely: single/double click, swipe up/down and back button functionalities. To the best of our knowledge, this is a first open-source system that provides this level of functionality for interfacing to an OS GUI.

Our research work leverages several open-source libraries, developed by a large community of computer vision researchers as well as skilled amateur programmers.

For our algorithm, we used the following libraries: openKinect [8], openNI [9], NITE and rgbdemo[10].

The rest of the paper is divided into the following three sections: background overview of default Kinect functionality and implementation of 3D hand tracking combined with activity recognition in Section 2; visual results showing usage of the system in Section 3 and Conclusion and future work in Section 4.

2. HCI Implementation using Kinect

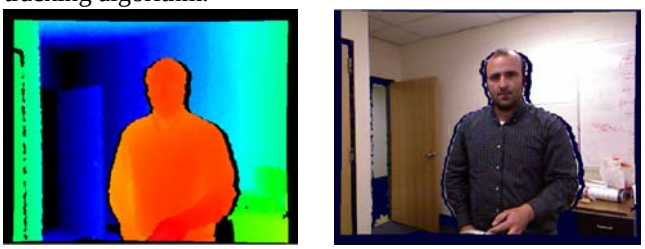
In this section we briefly describe the default Kinect functionality as well as our HCI implementation based on 3D hand tracking combined with activity recognition for OS GUI functionality.

2.1. Overview of Kinect sensor and built-in functionality

The Kinect sensor has an RGB video camera, and an active illumination IR light source separated by a baseline distance from a passive IR camera. The IR light source illuminates a view frustrum with time-coded structured light that is measured by a sensor chip which decodes the time of flight to recover range information on a per pixel basis. The 3D range image is augmented with RGB color information using previously determined camera calibrations. The end results is a RGB-Depth video image at 640x480 at 30fps, within a field of view of 57x43 degrees, with a dynamic 3D range of 0.8-10 meters. The range resolution varies with depth from 1.3mm at 0.8 meter to 5cm at 10 meters.

The RGB-depth information is streamed as two separate video frames over an USB connection. Using open-source driver and libraries, the data can be readily accessed with low, sub-30ms latency. Using previously determined camera calibration, along with a uint16-to-metric range scale factor, the two separate video streams can be combined into a single RGB-Depth (RGB-D) 3D point cloud. An example of the RGBD metric-upgraded 3D point cloud output is shown in Figure 1. From the video frames, we can readily determine features with a range resolution and angular resolution of about 1cm at 3 meters range.

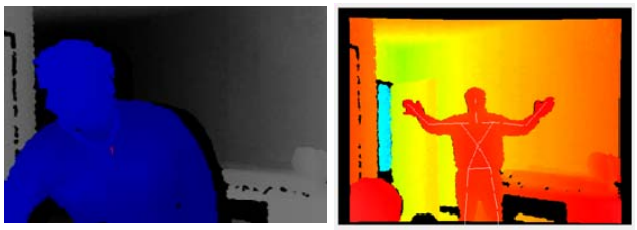
Aside from the raw RGB-D data, we leverage several open-source libraries, namely openKinect, openNI and NITE to determine human body pose. Figure 2 shows the detected body pose for single human subject as a skeleton overlayed in 3D using the NITE and rgbdemo libraries. After detection of the human body pose, our system now has the position of both hands, which act as location queues for the implementation of hand detection and tracking algorithm.



(a) (b)

(c)

Figure 1: a) Example of metric-upgraded 3D point cloud colored by z-depth in red-green-blue color scale. Red represents small z-depth, while blue represents large z depth. Black represents range nulls (no data). b) The same 3D point cloud now painted using RGB data, seen from Kinect camera perspective. C) Same RGB painted 3D point cloud now displayed from a different virtual camera perspective.



(a) (b)

Figure 2: a) Human detection. b) Pose estimation and corresponding skeleton overlaid in 3D to visually confirm that the correct 3D pose was found.

2.2. Hand on the fly Human Computer Interface

The first application is to remote control a computer

without a mouse. Examples of when this type of interface is useful include giving a slide and presentation, controller free remote control.

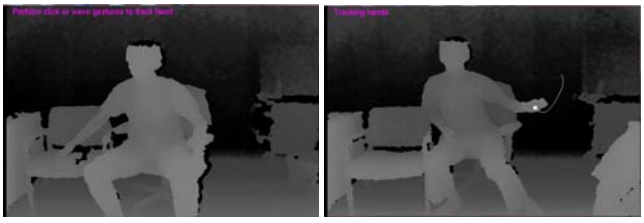
2.2.1 Hand Tracking for Cursor Navigation

Now that we have the body pose, we perform single hand detection followed by hand tracking. In our system, the hand tracking is analogous in GUI functionality to a computer mouse, in that our hand track direction moves the mouse in the same direction on the OS GUI, while the hand track velocity is mapped to a linearly proportional GUI OS cursor velocity.

To initialize the system to track a particular hand, we go into a mode where we are continually looking for what is known as "focus gesture," which acts as queue that we chose that particular hand. For the focus gesture to be detected, the user has to be within a range of 1 to 3.5 meters, with the hand not occluded by other objects.

There are two focus gestures that can be detected to start the initialization process: either "hand waving" or "hand clicking". The actions are detected by the presence of opposing 3D velocity vectors that add to a certain magnitude Mfocus(m/sec) within a short time window tfocus(sec).

Upon detection of the focus gesture, the hand is chosen for tracking. The algorithm for tracking is encapsulated within the NITE library and uses the 3D skeleton hand position to update the GUI xy pixel location. Because of the sensor noise due to hand shaking, there is a lot of noise embedded into the tracking location of the hand. We smooth the output x-y GUI position in time using a Gaussian time-window with tstd = 0.1 seconds. An example of the resulting smooth output hand trajectory is shown in Figure 3. Once we start tracking the hand, we now can move the GUI cursor. The relative movement of the GUI cursor in pixels is linearly proportional to the inter-frame relative movement of the hand in x-y Kinect camera location.



(a) (b)

Figure 3. Example of Hand Tracking. (a) Initialization with a focus gesture. (b) tracking hand with trajectory.

2.2.2 Gesture Detection

With the hand continuously tracked we enter a multi query mode where we asynchronously detect one of 3 major function classes:

1. Detect Click
2. Detect Steady
3. Detect Swipe

Figure 4 summarizes the gesture detection algorithm as a flow chart. The click function class is detected in a similar manner to the original focus gesture method: a click is detected by the presence of two video sub-frames with opposing range velocity vectors, adding to a certain magnitude large than Ms(m/sec) within a short time window tclick(sec). Within the function class, there are two subclasses, namely single-click and double-click. A single click is detected as opposing range velocity magnitude larger than Ms(m/sec), but smaller than Md(m/sec). A double click is detected when there are opposing range velocity vectors with magnitude larger Md(m/sec) within the time window tclick(sec).

The steady detection is used to allow for fine motor control, such as when you might need to click a small icon to minimize, maximize or close a GUI window. The "steady" class is detected if the moving time-window xyz location variance is small, below a certain std deviation pixel threshold Pstd with a time window tsteady. If we detect a steady action, the average GUI xy pixel location within the time-window is saved. The value can be used in conjunction with a click detection: if the current xy cursor position is within Pt of the saved steady-stage xy position, then the click will be registered at the steady-state position instead of at the current cursor position.

We also have a swipe class that is only enabled within a GUI window, and disabled when the GUI cursor is on the main desktop. The swipe class is detected as a fast movement in a constant direction of at least Sd within time tswipe followed by a low or zero velocity hand stop of at least Pstop, tstop to indicate that the GUI cursor should be stopped. The angular direction in the Kinect xy camera coordinates is binned into 3 sub-class quadrants to represent the following: from 45 to 135 degrees (i.e up movement) is mapped as a scroll up, from -45 to -135 degrees (i.e. down movement) is mapped as scroll down, for 135 to 225 degrees (i.e. left movement) is mapped as back button.

Once we have detected the function class and respective subclass, we map the actions to GUI commands. In our case, we tested the code under Microsoft Windows 7 and utilized the Windows API library to control the mouse, scroll up and down, press the back button and single or double click.

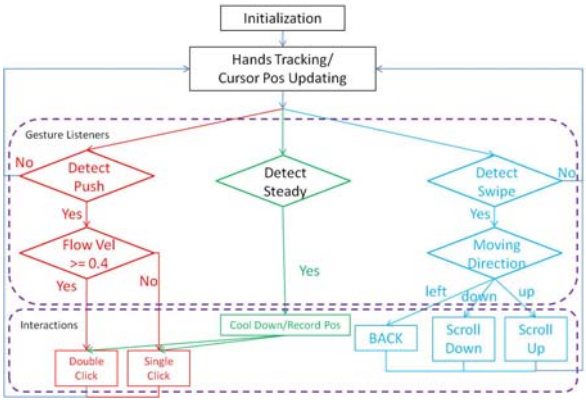


Figure 4: Overall working flow chart for the whole implementation.

2.2.3 Computer Interaction Parameter Definition

We defined the following parameters for our system. The focus-gesture time window tfocus=2 seconds with Mfocus=0.3m/s. The click function time window, tclick =1 seconds, with an single-click lower velocity detection threshold Ms = 0.2m/sec and a single-click upper velocity threshold Md = 0.4m/sec. The steady detection tsteady=2 sec, with pixel standard deviation of no more than Pstd = 25 pixels. The saved steady detected xy location will be used for click as long as the click is within proximity distance Pt = 50 pixels.

The swipe distance threshold was set to Sd = 0.25m within tswipe=0.3sec followed by a max delta position Pstop = 0.1m within delta tstop = 0.5sec.

2.3. Head-based Computer Human Interface

A second application was developed using a hands-free interface with similar functionality as the hand-tracking interface. The interface maps head movements to computer actions and is aimed at disabled people who might have reduced or no arm control.

2.3.1 Head Detection and 3D Pose Tracking

The head is detected using a HaarCascade frontal face detector [11] operating solely on Kinect color video data. An example of face detection is shown in Figure 4a.

After facial detection, head tracking in pixel-space is performed using a particle filter on the depth images [12]. We initialize the particle filter with a uniform distribution over 10 particles with standard deviation in both x and y direction, std_x and std_y = 10 (pixels); the standard deviation in width and height, std_w and std_h = 6 (pixels); the standard deviation of angle, std_a = 5 (degrees). One sample frame showing the position of the 10 particles is shown in Figure4b.

(a) (b)

Figure 4. Example Frames of Face Detection and Particle Filters. (a) Facial Detection with detected face marked by blue circle. (b) Particle filter with particles marked by blue squares with best particle marked by a red square.

2.3.2 Cursor Navigation

The system is initialized using 15 frames of RGB-depth data of a level head pose, with the corresponding cursor mapped to the center of the display. Once initialization is completed, the nose relative 3D position is tracked and compared to the initial position in order to move the cursor to a relative pixel offset on the display. To avoid head jitter and pose estimation error, the nose trajectory is smoothed using Gaussian filter with a std=0.1 seconds.

2.3.3 Action Detection and Implementation

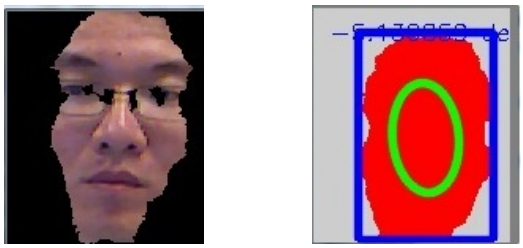
In order to detect various gestures using the head, we need to segment the head and determine its 3D pose. This is performed using both depth and color information to extract a facial RGB-depth image, as shown in figure 5a. Two categories of gesture recognition are defined as follows:

1. Steady location detection for precise click localization.
2. Head tilt detection for detecting single and double clicks.

A steady-location gesture is detected by aggregating the cursor position second-order statistics of 10 consecutive frames. If the std. deviation of the cursor is less than 25 pixels, a steady location gesture is detected; the mean cursor location is then stored to be possibly used for a follow-on click operation.

A click operation is defined as a head tilt exceeding 20 degrees off nadir. The tilt of the head is determined by constantly tracking the head pose using both color and depth data as shown in Figure 5b. Single left button click is mapped as a left-sided tilt, while right button click is mapped as right-sided tilt.

Thus, similar to the hand tracking interface, the hands-free interface provides cursor navigation functionality, as well as precise single left and right click operations.



(a) (b)

Figure 5. (a) Facial Region Extraction Sample (b) Facial Contour / Angle Detection Sample

3. Results and Discussion

In this section we will show that we can detect the gestures we described. Figure 6 attempts to visually demonstrate the detection of all the hand gestures described in Section 2. We demonstrate that we can navigate a Win7 OS GUI by clicking on different windows, scrolling up and down, enabling the return capability using a browser window, single and double clicking windows to minimize/maximize or close, and perform precise operations such as clicking on small buttons.

Further demonstration of hand tracking and gesture recognition can be found in the supplemental video presentation.

While our implementation works well in practice, there still is room for improvement. Here are some of the known issues to be resolved:

1. Hand shaking artifacts still pass through to the GUI, leading the cursor to not be stable and continue oscillating. More advanced filtering is needed to remove such noise.
2. Clicking after steady-state position finding sometimes is hard to achieve as the cursor moves too much due to movement of hand in not only range direction, but also camera xy direction. One solution would be to use the start position of the cursor movement before the click operation for comparison to steady-state position, instead of cursor position after. Need to implement cursor buffer to achieve this objective, which requires more coding.
3. When hands are at the camera boundary, the cursor cannot traverse the whole GUI screen. We lack a method to stop hand tracking, which would be the equivalent of picking up your mouse to reposition it on the mouse pad to continue traversing the GUI screen without moving further in the mouse physical world coordinate space.
4. Click behavior is sometimes not detected to generate click events.

4. Conclusion and Future Work

In this work we explored an HCI application using the Kinect sensor to capture human gestures that were then translated to OS GUI actions. We leveraged off previous libraries and default Kinect functionality for detecting human subjects/pose to extend HCI functionality for cursor movement and gesture recognition. This functionality is the first of its kind that is open source and available for the whole user community.

Cursor movement functionality was implemented by tracking a human subject hand in real time. Furthermore, gesture recognition was recognized with low latency. Our system could detect 3 main gesture classes, namely, click, steady and swipe. In addition, sub-classes of gestures were detected such as single/double click, swipe-scroll-up, swipe-scroll-down, swipe-back, for a total of 6 gestures.

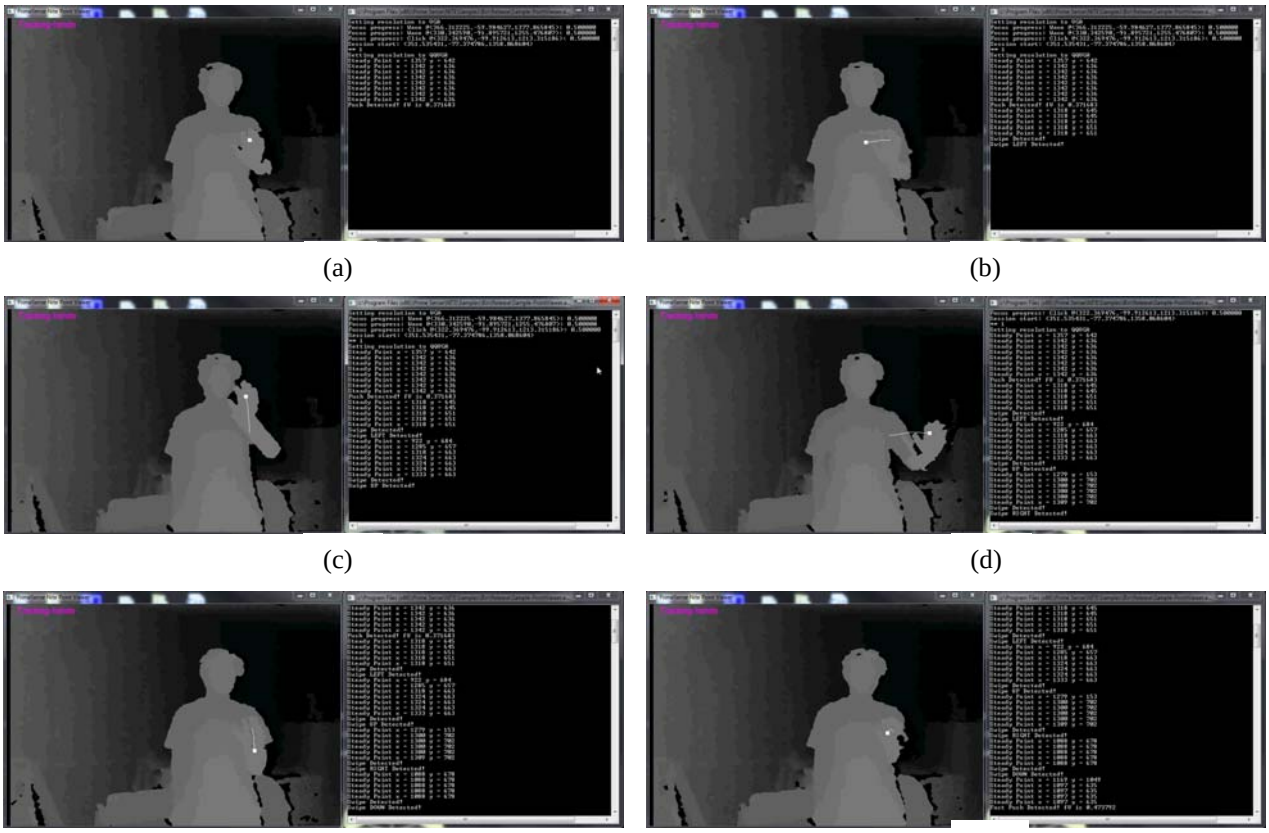
These gestures were demonstrated for navigating a Win7 OS GUI by clicking on different windows, scrolling within a window, minimizing/maximizing/closing a window and precise button clicking on small buttons. A further real time demo will be given to showcase our system's capabilities.

We plan to release the code as open source under the GNU license in the hope that it will be useful for the whole community. Further potential research will be to develop multi-hand tracking in order to define and detect a multitude of different gestures to better control the OS GUI. Other applications of interest are large-scale indoor mapping, human activity recognition using 3D RGB-D data, surveillance and robotic exploration.

References

[1] http://www.apple.com/macosx/whats-new/gestures.html
[2] https://researcher.ibm.com/researcher/view_pic.php?id=147
[3] Tanriverdi, V., & Jacob, R. J. K. (2000). Interacting with eye movements in virtual environments. In Human factors in computing systems: CHI 2000 conference proceedings (pp. 265-272). New York: ACM Press.
[4] Starker, I., & Bolt, R. A. (1990). A gaze-responsive self-disclosing display. In Human factors in computing systems: CHI '90 conference proceedings (pp. 3-9). New York: ACM Press
[5] Jacob,R. J. (1990).What you look at is what you get: Eye movement based interaction techniques. In Human factors in computing systems: CHI '90 conference proceedings (pp. 11-18).New York: ACM Press.
[6] A.T. Duchowski, A breadth-first survey of eye tracking applications, Behavior Research Methods, Instruments, and Computing **34** (4) (2002), pp. 455–470.
[7] http://www.vicon.com/
[8] http://openkinect.org/wiki/Main_Page
[9] www.openni.org
[10] http://nicolas.burrus.name/index.php/Research/KinectRgbDemoV5?from=Research.KinectRgbDemoV4

[11] http://opencv.willowgarage.com/wiki/FaceDetection
[12] http://code.google.com/p/opencvx/



(a) (b) (c) (d) (e) (f)

Figure 6: Different gesture detections under our implementation. The left side of each image is the depth image from Kinect with hand tracking trajectory. The right side is the detection results output. (a) Push Detection (b) Swipe Left Detection (c) Swipe Up Detection (d) Swipe Right Detection (e) Swipe Down Detection (f) Fast Push Detection.