Using Microsoft’s Kinect to Interpret Gestures and Control a Personal Computer

Albert Gural

TJHSST
COMPUTER SCIENCE RESEARCH LAB

June 13, 2012
Using Microsoft’s Kinect to Interpret Gestures and Control a Personal Computer

Albert Gural
TJHSST
COMPUTER SCIENCE RESEARCH LAB

Abstract—Various input devices exist for controlling the computer, including keyboards and mice. While these devices are good at most tasks, they can be unintuitive for more manipulative-heavy tasks, such as CAD. Mouse control in many CAD programs is often clumsy as movements on a 2D surface must translate to controls in a 3D virtual environment. This method works, but it is far from the most efficient or intuitive input system. My project aims to provide an alternative, more intuitive approach to input – gesture input.

I. INTRODUCTION

HAND GESTURE USER INTERFACES are an emerging technology, with applications ranging from home use to large-scale systems. While mouse-based user interfaces are acceptable means of controlling an environment, gesture-based UIs could make controlling certain environments more intuitive [7]. In fact, many vendors have already shipped out devices with gesture controls. For example, many touch screen smartphones rely on touch gestures to operate the OS. There are also many other similar systems for larger scale applications [6].

My aim with this project is to extend on a primitive gesture tracking system to 3D hand-gesture tracking for input to a computer. While this gesture-based UI is not necessarily new technology, current implementations are weak and tend to cause more problems than solutions. My solution involves using Microsoft’s Kinect to track the 3D alignment of fingers in order to control an environment, allowing improvements to the usability of certain applications.

A. Fingertip Tracking with the Kinect

Currently, Microsoft has an API which is already being utilized for game control of the Xbox using the limbs of users.

However, this API does not allow for fingertip detection, presumably because of the lack of camera resolution and difficulty of tracking such small objects. Because my application revolves around hands being close to the sensor, the sensor would in theory be able to detect fingertips with reasonable accuracy.

II. PURPOSE

The main purpose of this project is to provide an alternative means of controlling a virtual environment
than the standard keyboard/mouse. Hand gesture-based UIs have the potential to provide a quicker and more intuitive input mechanism. Additionally, some tasks may be completely trivialized by the use of hand-gesture interpretation, such as 3D modeling.

III. PRIOR WORK

A. Contrast Point Gloves

This solution consists of a wearable glove with bright dots, known as contrast points. These contrast points let the camera easily determine the fingertip positions. This solution works well and can be very quick, but wearing glove can be inconvenient for users.

B. OpenCV Processing

Images from a color camera are processed with OpenCV to find approximate hand contours, however, this requires advanced hand detection. While relatively simple hardware are needed (and there’s no need for a high contrast point glove), the requirement of complex algorithms causes these solutions to generally be unreliable and slow.

IV. BASE SYSTEM

A. Operating System

1) Linux: At first, I considered using a Linux distribution as the OS; specifically, Ubuntu 10.04. MIT’s media lab produced a system for Kinect-input processing, using their MIT-ROS base [10]. I found that the system, while it covered most of the functionality I was interested in, was a bit too bulky. The install itself was over a gigabyte and the installation process was probably too tedious for most average users.

Other systems designed for Linux distributions, such as openNI, were also not easy to set up [5] [8].

2) Windows 7: The Kinect was originally designed for use on an Xbox or a computer running a Windows OS. Thus, setup on Windows can be a bit simpler. Microsoft provides a Kinect API, however this system doesn’t support fingertip detection [4].

The solution I decided to use as the base was developed by Robert Walter of the Berlin Institute of Technology. He created a system known as FingerTuio3d that fairly successfully determines fingertip positions [9].

V. FINGER TUIO3D

A. Methodology

FingerTuio3d works off of a four-step process after retrieving the initial image from CL NUI (Windows Kinect driver):

1) Depth Thresholding - Only elements within a certain depth are considered, since hands generally fall within a certain depth of the sensor.
2) Contour Extraction - OpenCV determines contours of the image objects. Each contour is a vector of point2d that gives points along the contour.

3) Approximate Contours - Points along the contour are used to determine a convex hull of the hand/arm system.

4) Fingertip Determination - This assumes vertices of the convex hull to be fingertips if their interior angle is less than 1 radian, and the points fall within the top 90% of the contour.

B. Installation

I used the following steps to set up FingerTuio3d.

1) Download and Install the following prerequisite items:
   - Download and extract http://dl.dropbox.com/u/5505209/FingertipTuio3d.zip
   - Download and extract OpenCV 2.1 for Windows to Program Files
   - Download CMake
   - Microsoft Visual Studio 2010 for Visual C++

2) Follow the following guide to get OpenCV working in VS 2010:

3) Open FingertipTuio3d.sln in VS 2010

4) Apply the project includes for OpenCV. Includes for CL NUI:
   - Add to Includes:
     C:\Program Files\CodeLaboratories\CL NUI\Platform\SDK\Include
   - Add to Libraries:
     C:\Program Files\CodeLaboratories\CL NUI\Platform\SDK\Lib

Windows dlls - add to Windows Path:
   a) C:\Program Files\CodeLaboratories\CL NUI

5) Working on Project
   a) Open FingertipTuio3d.sln in VS 2010
   b) Change line 146 to
      CLNUICamera cam = CreateNUICamera(
                      GetNUIDeviceSerial(0));
   c) Press F7 to compile and F5 to run

VI. Procedure

The main procedure consists of several overall steps:

1) Input Image
2) Contour Processing
3) Fingertip Detection
4) Gesture Determination
5) Environment Control

The first three procedure goals are based entirely off of the FingertipTuio3d work by Robert Walter. I examined the process, experimenting on ways to improve fingertip detection. Additionally, I looked at implementing gesture detection.

A. Input Image

The image input comes directly from the Kinect as a pair of stereoscopic images. The Kinect API can do some additional processing to reduce the data to skeletal-tracking. However, support for finger tracking is not supported, so OpenCV processing is necessary to determine fingertip coordinates.

B. Contour Processing

We’re only interested in objects within a certain range of depths, since hands will in general fall within a certain distance from the sensor. After filtering out parts of the image outside a certain depth threshold, object contours are determined. Contour determination of the hand/arm regions is done with OpenCV::findContours. This function locates regions bounded by image edges and returns a vector of contours, each a vector of Point2d.
C. Fingertip Detection

For a hand that is held out, fingertips will generally be on vertices of a convex hull (convex bounding polygon). Thus, fingertip detection is based on examining only points along the convex hull. This system works in cases where a hand is fully or partially extended.

In order to distinguish between finger and non-finger points on the convex hull, the angle of the convex hull points to other convex hull points is considered. Only points with angles less than 1 radian are considered. Additionally, hands generally occur with fingers on top, so only points in the top 90% are considered.

D. Gesture Determination

This process involves determining the intended gesture based on the series of input hand skeletons.

1) Static Gestures: For each input frame, the static gesture is determined by finding a “best fit” gesture (the gesture whose coordinates differ the least from those of the input frame. Optimality is determined by taking the magnitude of the difference of the two vectors of coordinates.

In the example above, the pixel-coordinates of each point are given (row, col). Two sets of these points can be compared by summing the square of the difference of every set of mapped points. This value is the “distance” between the two sets of points. The smaller the distance, the closer the match. The two sets of static gestures I looked at are described in subsection Environmental Control.

2) Dynamic Gestures: Next, dynamic gestures are determined by looking at the sequence of images to determine which fingertip points are moving the most, as well as the speed at which they’re moving. Once the “moves” are determined, this information must be
processed to determine the intended gesture.

One dynamic gesture is considered. When a closed fist is detected, the start coordinates are recorded. The hand is subsequently tracked, and at each frame, the absolute movement of the hand is determined. The system stops tracking the hand once it stops being in the closed fist position.

E. Environmental Control

Once the gesture is determined, it has to be interpreted as an intended control mechanism. The main issue is determining a set of useful controls, as well as how to activate them (what gesture to use). There are a wide range of possible gestures, but for my project, I looked at two simple forms of gestures.

1) Hand Position: This involves looking at the actual position of each hand; determining whether a hand is on the left or right side of an user. To do this, I compared a collection of points comprising the fingertip positions to a test collection for a sample “right” and sample “left” hand. The magnitude of the difference of the two vectors of points gave which side the hand was more likely on.

2) Open or Closed Hand: The process for this was similar to that of looking at the hand position. However, an added complication came up from fingertips disappearing under the closed hand, due to fingertips moving inside the convex hull. This was fixed by looking at each possible mapping from input joints to library gesture points, then picking the minimum of those as the final “distance” between the input image and the given library gesture.

An open hand was easy to determine because all fingertips were on the convex hull.

However, a closed hand was difficult to determine due to lack of fingertips. Additionally, the small size of the object caused it to fall under the object size threshold.

I found that a quicker and more reliable way to sense closed hands as opposed to open ones was to look at the number of detected fingertips. When the number falls below a certain threshold, it is considered a fist.

VII. Results and Conclusion

The robustness and range of capabilities of the system could use a lot of work, but my project showed the viability of such a system. Detection of the fingertip points was done quickly enough for real-time detection on a relatively underpowered machine (1.6 GHz Intel Core i5 processor).

In terms of accuracy, the fingertips could be consistently tracked across the screen. I got accurate depth data and found the fingertip locations in the image for a pair of spread out hands. I was able to determine the best-fit gesture for very simple actions - finding hand locations and determining whether a hand is open or closed. This data, combined with depth data, could allow for simple CAD manipulations such as global rotation.
VIII. FURTHER RESEARCH

If time permits, the following would be interesting extensions to this project.

A. Robustness

Because of the reliance on convex hull points, fingers that were not completely extended were not visible. Additionally, for a closed hand, there was not enough contrast in the depth data to demarcate the contour, and in a rotated hand, fingertips were obscured from vision.

B. Specific User Calibration

Adding special calibration for specific users would allow it to be more accurate for a wide variety of people. This could be accomplished simply with a calibration pose (both hands stationary in the air).

C. Specialized Software

Many features could be enhanced by designing software specifically around taking input directly from this program, instead of indirectly controlling a program through Windows events.

D. Extending to Use with Single Camera

While using the Kinect was the main goal of this project (taking advantage of the stability two cameras provide), extending this to only requiring one camera would be an important goal. Not only would this be considerably cheaper, it would also allow for operation via the provided web camera that is standard in most laptops.

REFERENCES