Human-Computer Interaction using Gesture Recognition and 3D Hand Tracking

Jakub Segen and Senthil Kumar
Bell Laboratories
segen@lucent.com, senthil@bell-labs.com

Abstract

This paper describes a real time system for human-computer interaction through gesture recognition and three dimensional hand-tracking. Using two cameras that are focused at the user's hand, the system recognizes three gestures and tracks the hand in three dimensions. The system can simultaneously track two fingers (thumb and pointing finger) and output their poses. The pose for each finger consists of three positional coordinates and two angles (azimuth and elevation). By moving the thumb and the pointing finger in 3D, the user can control 10 degrees of freedom in a smooth and natural fashion. We have used this system as a multi-dimensional input-interface to computer games, terrain navigation software and graphical editors. In addition to providing 10 degrees of control, our system is much more natural and intuitive to use compared to traditional input devices. The system is user independent, and operates at the rate of 60 Hz.

1 Introduction

Hand gestures provide a very convenient and natural way of interacting with the computer for many applications. Many actions such as terrain navigation are naturally expressible in terms of hand gestures rather than key-strokes or joystick movements and it would be more appropriate to input such actions to a computer using hand gestures. For example, pointing with a finger is a more natural and intuitive way to specify direction of motion (in 3D) than key-strokes or joystick movements. The use of hand gestures as input interface is addressed in several recent publications [5, 4, 7, 1, 8, 9, 12, 10, 6, 11]. For example, Maggioni [7] describes a system with two cameras that can recognize six static gestures. This system can also compute the position of the palm in the image and its distance to the camera. In [4] and [5] Kjeldsen and Kender describe a system that uses hand tracking and recognition to replace the mouse for certain actions like moving and resizing windows. In [2] Huang and Pavlovic present a review of recent work on gesture recognition and hand tracking.

This paper describes our on-going research on human-computer interaction through real time gesture recognition and three dimensional hand-tracking. Using two cameras that are focused at the user's hand, the system recognizes three gestures and tracks the hand in three dimensions. The system can simultaneously track two fingers (thumb and pointing finger) and output their poses. The pose $(X, Y, Z, \alpha, \epsilon)$ for each finger consists of three positional coordinates and two angles (azimuth and elevation). By moving the thumb and the pointing finger in 3D, the user can control 10 degrees of freedom in a smooth and natural fashion. We have used this system as a multi-dimensional input-interface to computer games, terrain navigation software and graphical editors. The two fingers can also be used as a virtual robot arm allowing the user to grasp and move virtual objects in 3D (see Figure 1). In addition to providing 10 degrees of control, our system is much more natural and intuitive to use compared to traditional input devices. The system is user independent, and operates at the rate of 60 Hz.

Our system is meant to augment (and, in some cases, even replace) devices such as the computer mouse, joystick, or track-ball. The main advantages of our system compared to the traditional input devices are as follows:

- With input devices such as the mouse, joystick or tablet that have only two degrees of freedom, several consecutive actions are needed to control more than two parameters. Such control is not always natural and requires much user training. In our system, a command with up to 10 arguments can be issued with one gesture. For example, one can use the hand in a very natural fashion to fly through a virtual three dimensional scene while simultaneously controlling the position (X, Y, Z) and orientation (roll, pitch and yaw).
- In our system there is no need for any device in

0-8186-8821-1/98 $10.00

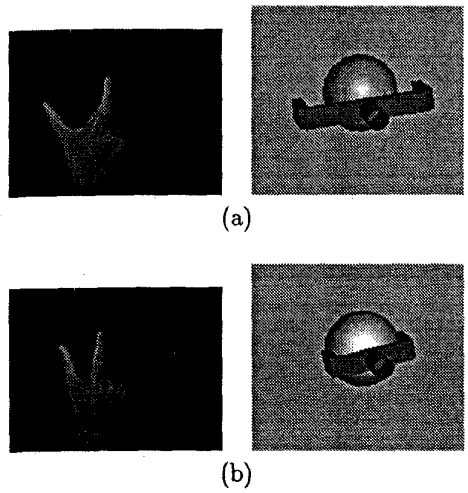


Figure 1: Driving a robot arm with gestures. See Section 3.

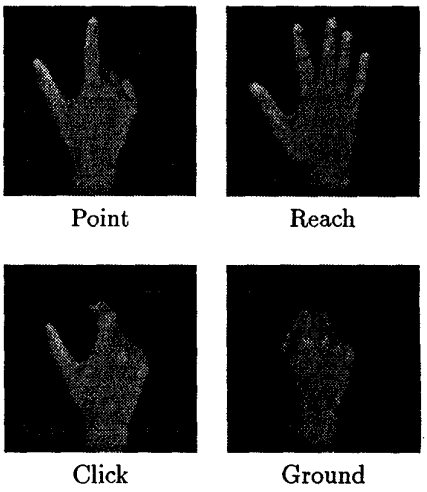


Figure 2: Gestures Recognized by the system.

the immediate neighborhood of the user except for a pair of video cameras which can be several meters away.

Our system uses two cameras placed several feet above the desktop and several feet apart such that they can both see the user's hand. The cameras are calibrated with respect to a world coordinate frame and gen-locked together. The processing is divided into two stages - the first extracts 2D information from the two images and the second combines the information from each image to obtain 3D information about the thumb and the pointing finger.

2 Hand Tracking and Pose Estimation in the Image Plane.

At each time instant, the system receives two images (one from each camera) of the scene and classifies them as belonging to one of three gesture classes. These classes are (Figure 2): (1) *Point* : User is pointing with a finger. (2) *Reach* : All the fingers are stretched out. (3) *Click* : Quick bending of pointing finger. A gesture that is not recognized is classified as *ground*. The various stages of the algorithm are explained below.

Region Extraction and Perimeter Screening

Connected regions are extracted by comparing the input image with a previously acquired background image. After extracting the regions, the boundary of each region is represented as a list of pixel positions $\{(x(i), y(i))\}$, in a clockwise order. A heuristic screening of the regions determines which one represents the hand. A region is assumed to be a hand if its perimeter length falls between two fixed thresholds T_1 and T_2. If exactly one region satisfies the above rule the method proceeds to the next step, otherwise it returns "ground", to signal an unrecognized gesture.

Computation of Local Features

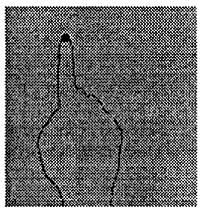
The region boundary is represented as a sequence of boundary points $P(i) = (x(i), y(i))$ and the k-curvature is measured at each point. The k-curvature is the angle $C(i)$ between the vectors $[P(i-k), P(i)]$ and $[P(i), P(i+k)]$, where k is a constant. The points along the boundary where the curvature reached a local extremum are identified as "local features". Some of these local feature are labeled "peaks" or "valleys". Peaks are those features whose curvatures are positive (denoting a locally convex boundary) with magnitudes greater than a fixed threshold P_{thr}. Valleys are features whose curvatures are negative (denoting a locally concave boundary) with magnitudes less than a fixed threshold V_{thr}.

Gesture Classification

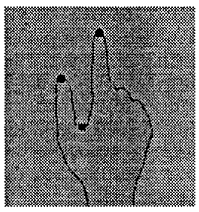
This step first performs a preliminary gesture classification based on the number of peaks (N_{peaks}) and the number of valleys ($N_{valleys}$) and stores the result in a variable *class*. The rules are of the type:

$$\begin{array}{lll} point & : & (N_{peaks} < T_p) \\ reach & : & (N_{peaks} > T_{r1}) \ or \ (N_{valleys} > T_{r2}) \end{array} \quad (1)$$

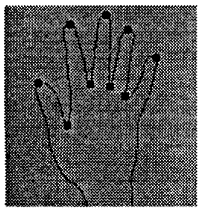
Figure 3 illustrates the motivation behind this technique. In the figure, peaks and valleys are denoted by



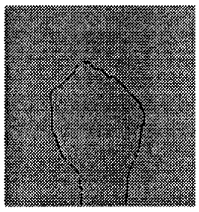
Point (1 peak, no valleys)



Point (2 peaks, 1 valley)



Reach (5 peaks, 4 valleys)



Ground (no peaks, no valleys)

Figure 3: The numbers of peaks and valleys is used for preliminary gesture classification. Circles represent peaks and squares denote valleys.

solid circles and squares. The result of this step is called *class*. It can be either *point*, *reach*, or *ground*.

This preliminary classification (*class*) is then passed to a finite state classifier which determines the gesture. The type of gesture is stored in a variable *state*, which takes one of four values: *ground*, *reach*, *point*, and *click*. The new value of *state* depends on the previous value of *state* and additional analysis of the boundary. The classifier proceeds as follows.

If *class* is not *point* then it is returned as the new *state*. If *class* is *point*, the pointing finger is found and the pose is computed, returning *ground* if it fails. If previous value of *state* is either a *point* or a *click*, the test for a *click* is done, setting *state* to *click* if it succeeds or to *point* if it fails. If the previous value of *state* is *reach* or *ground*, *state* is set to the same value as *class*.

At termination, the variable *state* describes the gesture. If the gesture is *point* or *click* the variable *pose* contains the pose (x, y, θ) of the pointing finger.

Detection of Pointing Finger and Thumb

This step is executed only if the variable *class* has the value *point*. If the number of peaks is one, that peak is returned as corresponding to the pointing finger. If there are two peaks (see Figure 3), the one on the "right" is returned as corresponding to the pointing finger and the other as corresponding to the thumb(this applies for the right hand, for the left hand, the peak on the left would correspond to the pointing finger). To determine that one peak is to the "right" of another we proceed as follows.

The boundary is represented as a sequence of (x, y) coordinates ordered clockwise. Let the size of this list be N. Let the two peaks be P and Q and their indices in the above sequence be p and q (*i.e.* P is the p_{th} point and Q is the q_{th} point in the list). We define $\mathbf{cnorm}(\mathrm{x})$ as:

$$\mathbf{cnorm}(x) = \begin{cases} x + N & : \quad x < 0 \\ x - N & : \quad (x+1) > N \\ x & : \quad Otherwise \end{cases} \tag{2}$$

$\mathbf{cnorm}(p - q)$ gives the number of points between P and Q traveling clockwise along the boundary from Q to P. If $\mathbf{cnorm}(p - q) < N/2$ then P is to the right of Q.

Pose computation

The previous step determined the indices of the pointing finger and the thumb (if visible). The *Pose Finder* uses this information to determine the orientation of the finger.

Let i_f be the index of a finger tip along the contour and let $P(i) = [x_i, y_i]^T$ be the i_{th} point along the hand's contour. Then $P(\mathbf{cnorm}(i + k))$ denotes a point that is k points away from $P(i)$ to its right along the hand's contour and $P(\mathbf{cnorm}(i - k))$ denotes a point that is k points away from $P(i)$ to its left. A mid point $Q(k)$ is computed as

$$Q(k) = \frac{P(\mathbf{cnorm}(i_f + k)) \; + \; P(\mathbf{cnorm}(i_f - k)}{2}. \tag{3}$$

The *Pose Finder* computes $Q(k)$ for $k_{min} < k < k_{max}$ where k_{min} and k_{max} are constants. A line L is then found by least squares fit to the midpoints $Q(k)$. (see Figure 4). The orientation θ of L gives the orientation component of the pose. The position is found as the intersection of L with the boundary.

Click Detection

We define the *click* gesture as "stop-and-click", rather than a "running click". In other words, during the *click* gesture, the dominant motion is only from the pointing finger - the rest of the hand remains more or less motionless. The *Click Detector* classifies a gesture as *click* if the whole hand has not significantly moved from the previous frame and the position component of pose has shifted by a specified amount towards the interior of the region. The motion of the hand is measured using a set of reference points along

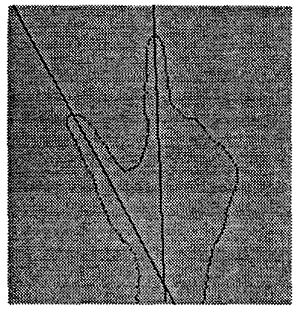


Figure 4: Pose computation.

the hand boundary selected outside the pointing finger.

3 10 DOF Pose

For each visible finger, the 2D analysis stage of our algorithm gives us a set of three numbers (x_1, y_1, θ_1) from the first image and another set of three numbers (x_2, y_2, θ_2) from the second image. These six numbers are combined to produce the five quantities $(X, Y, Z, \alpha, \epsilon)$ which comprise the 3D pose of the finger.

Given the 3×4 perspective projection matrices $\mathbf{M_1}$ and $\mathbf{M_2}$ of the two (calibrated) cameras, (X, Y, Z) can be computed as the intersection of two lines [3]. To determine the orientation of the finger's axis, define the vectors $\mathbf{u_1}$ and $\mathbf{u_2}$ by $\mathbf{u_1} = [cos(\theta_1),\ -sin(\theta_1),\ y_1 sin(\theta_1) - x_1 cos(\theta_1)]^T$ and $\mathbf{u_2} = [cos(\theta_2),\ -sin(\theta_2),\ y_2 sin(\theta_2) - x_2 cos(\theta_2)]^T$. Then, the finger's axis in the first image is represented by the line $u_{11}x + u_{12}y + u_{13} = 0$ and that in the second image by $u_{21}x + u_{22}y + u_{23} = 0$. The axis of the pointing finger in the world coordinate system is the intersection of two planes whose parameters are given by the vectors $\mathbf{u_1'}$ and $\mathbf{u_2'}$ where $\mathbf{u_1'} = \mathbf{M_1^T u_1}$ and $\mathbf{u_2'} = \mathbf{M_1^T u_2}$. The azimuth and elevation angles are computed from the parameters of the axis. Applying the above procedure to the pointing finger and the thumb gives us a total of 10 parameters.

4 Performance and Accuracy

The system operates at the rate of 60 frames/sec on two 200 MHz SGI Indy computers, with a load of about 30% on each cpu. The only reason we need two machines is that the Indy takes only one 60 frames/sec video stream. The computer-code should run at the same rate on a single machine as long as we can provide two 60 frames/sec video inputs.

Our system is user independent and the three gestures are recognized almost perfectly. We performed experiments with users whose age, hand size and skin complexion varied. The users were asked to maintain the same gesture as they moved their hand in 3D. The recognition error rate is less than 1/500 with slightly higher rates when the normal to plane of the hand made large angles with the optical axis of either camera. The stability and accuracy of the 3D pose depends both on the accuracy of the estimated planar pose and the accuracy of the camera calibration. To measure the stability of the pose estimation, we conducted a number of experiments where the pose was computed while the hand was held steady. The estimated planar pose is very stable with the jitter in position being less than 1 pixel and the jitter in θ being less than half a degree. The jitter in 3D pose varied with the position and orientation of the hand. Typically the jitter in orientation is less than 2 degrees and the jitter in position is less than 1 to 5 millimeters.

5 Applications

We have used our gesture recognition and hand tracking system as an input interface to a number of applications that require multi-dimensional control. These include computer games, virtual flight simulators and 3D design tools. We describe three of these applications below.

The first application is a 3D Scene Composer that allows the user to produce complex 3D scenes by selecting simple primitives from a menu and manipulating them. An example is shown in Figure 5. Here the user controls the robot arm by moving the thumb and the index finger. The distance between the two grippers depends on the distance between the two fingers. The fingers also control the position and orientation of the robot arm. All operations in this system (including menu selection) are gesture driven.

The second application (Figure 6) involves virtual fly-thru's over terrains. Here, the user stretches out the thumb and the pointing finger and imitates flying by moving the hand. The roll, pitch and yaw angles of the flight are controlled by the corresponding angles of the hand. The third application is an interface to the popular game Doom. Here the user navigates corridors by finger-pointing. In addition, the user can fire the gun using the "click" gesture and open doors with the "reach" gesture.

6 conclusions

We presented a gesture based system for human-computer interaction. The system recognizes three gestures and computes up to 10 parameters. The sys-

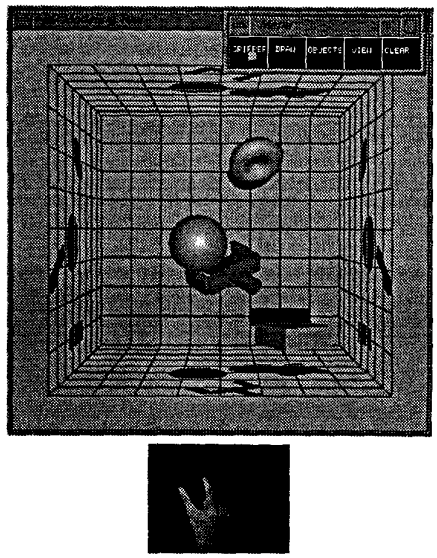


Figure 5: Scene Editor.

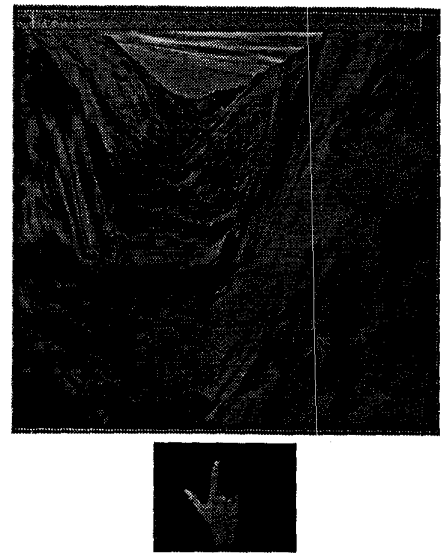


Figure 6: Fly-thru.

tem is user independent and robust. Users who tried our interface found it much more natural and intuitive and, liked not being bound by wires and mechanical devices.

References

[1] A. Azarbayejani, T. Starner, B. Horowitz, and A. Pentland. Visually Controlled Graphics. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 15(6):602–605, June 1993.

[2] T. Huang and V. Pavlovic. Hand Gesture Modeling, Analysis and Synthesis. In *Proc. International Conference on Automatic Face and Gesuture Recognition*, pages 73–79. June 1995.

[3] R. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision.* McGraw Hill, 1995.

[4] R. Kjeldsen and J. Kender. Visual Hand Recognition for Window System Control. In *Proc. International Conference on Automatic Face and Gesuture Recognition*, pages 184–188. June 1995.

[5] R. Kjeldsen and J. Kender. Towards the use of Gesture in Traditional User Interfaces. In *Proc. International Conference on Automatic Face and Gesuture Recognition*, pages 151–156. October 1996.

[6] M. W. Krueger. *Artificial Reality II.* Addison-Wesley, 1991.

[7] C. Maggioni. GestureComputer - New Ways of Operating a Computer. In *Proc. International Conference on Automatic Face and Gesuture Recognition*, pages 166–171. June 1995.

[8] J. M. Rehg and T. Kanade. DigitalEyes: Vision Based Human Hand Tracking. In *CMU Tech Report CMU-CS-93-220.* 1993.

[9] J. Segen. Controlling Computers with Gloveless Gestures. In *Proceedings of Virtual Reality Systems.* 1993.

[10] V. J. Vincent. Dwelving in the depth of the mind. In *Proc. Interface to Real and Virtual Worlds.* 1991.

[11] D. Weimer and S. K. Ganapathy. Interaction Techniques using Hand Tracking and Speech Recognition. In *Multimedia Interface Design, ed. M. Blettner and R. Dannenbergc*, pages 109–126. Addison-Wesley, 1992.

[12] P. Wellner. The DigitalDesk Calculator: Tangible Manipulation on a Desktop Display. In *Proc. ACM Symposium on User Interface Software and Technology.* November 1991.