A Usable Real-Time 3D Hand Tracker

Subutai Ahmad
Interval Research Corporation,
1801-C Page Mill Rd., Palo Alto, CA 94304
ahmad@interval.com

Abstract

This paper presents a computer vision system for tracking human hands. The algorithms used to extract the 3D position and planar orientation of the hand, and the joint angles of the fingers are described. The combined system is able to track a natural hand at 30 frames per second on a standard workstation with no special image processing hardware other than a frame grabber. The tracker has been used as an interface for navigating around virtual worlds.

1 Introduction

Is it feasible to build a practical and non-intrusive hand tracking device? The answer is at least a partial "yes". In this paper I give an overview of two working systems, based on computer vision, for extracting the 3D position, planar orientation, and the finger joint angles of natural hands.[1]

The primary motivation of this work has been the development of an ideal interface for manipulating three dimensional objects and for navigating around three dimensional environments. It is clear that our hands are our primary means of interacting with our physical environment. As such our hands have evolved to be versatile but highly complex devices. In fact, as children, we spend years mastering their use. Rather than invent completely new interfaces it is natural to take advantage of this intense training and attempt to exploit the way we use our hands.

There are a number of stringent requirements that must be met in order for a hand tracking system to be really useful as an interface. First, real time performance is critical. In our experience we find that the latency must be smaller than 100 milliseconds for the system to be bearable, and should really be smaller than 50 msecs for extended use. This agrees qualitatively with the psychophysical literature on "the psychological moment"[6] (briefly: visual events that occur within 100 milliseconds are naturally integrated, events that occur outside that time window are not). Second, we must use technologies that keep the user as unencumbered as possible. The system should be able to extract the desired information without requiring the user to wear gloves, wires, or other encumberances as with the DataGlove[9].

[1] Gesture recognition, or the task of interpreting the hand configuration, is beyond the scope of this paper. See [5] for a good introduction to this topic.

These requirements pose challenging problems for computer vision. No general purpose solutions are known. The rest of this paper describes some attempts at solving the task outlined above. The approach is quite specific to hand tracking but will hopefully lead to insights that are more generally applicable.

2 A Four Degree of Freedom Hand Tracker

We now describe the first version of our system, implemented in 1992 at Siemens Central Research, Munich. This system is able to reliably track a normal (unmarked and unencumbered) hand in real time. Unlike [4, 5] the system is relatively insensitive to image clutter and extracts three dimensional positional information. This version of the tracker can run on a standard Sun Sparcstation 10 at a speed of 30 frames per second, without any special image processing hardware other than a framegrabber. (A previous version of the system required a person to wear a specially marked cotton glove[8].) There are essentially two parts to the system: a segmentation module and a control strategy. These are described in turn below.

2.1 Color Histogram Based Segmentation

Given sequences of camera images, an important task of the system is to separate the target hand from the background. The main challenge is to build a segmentation module that is insensitive to normal backgrounds and that operates in real-time. This is not an easy task as the image may contain many irrelevant and confusing details (See Figure 1).

Our algorithm works by estimating the distribution of skin colors on a person's hand. This information is then used to detect which parts of the image belong to the hand and which are part of the background. The hope is that as long as the objects in the scene do not have the same color distribution as the user's hand the segmentation will be relatively clean.

2.1.1 Constructing the Histogram

A color-histogram is used to estimate the distribution of colors in a patch of skin. The basic idea behind a color histogram is to partition RGB color space into a number of bins. To train the system we select a patch of skin from an image (by a "patch" we mean a small square region in the image). From the set of

1058-6393/95 $4.00 © 1995 IEEE

Figure 1: Example of a typical image with a hand. Note the complex background which the system has to ignore.

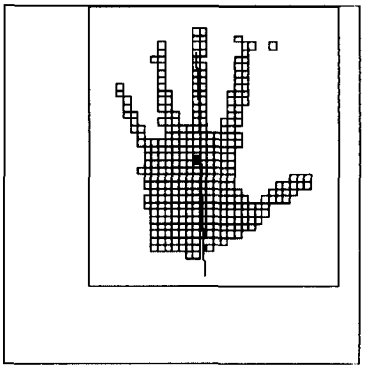

Figure 2: The result of segmenting a hand with a color histogram. Each small unfilled rectangle represents one segmented patch. The filled square represents the center of mass, the line represents the hand's orientation, and the inner large rectangle the search region for the next frame.

pixels in this patch, a color histogram is constructed by counting the number of pixels that fall into each bin. By computing a histogram of sample patches from the skin in this way, the system essentially constructs a rough estimate of the probability distribution of colors in the skin. Instead of RGB space we use a normalized 2D space that tends to eliminate the effects of varying illumination. Given an example skin-colored pixel, with color values r, g, and b, we compute its normalized colors as $r' = r/(r+g+b+1)$ and $b' = b/(r+g+b+1)$. These give values between 0 and 1 and can be thought of as computing the percentage of red and percentage of blue. Each of the dimensions r' and b' are discretized by a factor of d. The histogram is then a d x d array.

2.1.2 Histogram Based Segmentation

At run time the tracking system repeatedly matches small patches of the image to the stored histogram using a matching algorithm. To match two histograms, the histogram intersection algorithm [7] is used. The match score $\mathcal{M}$ between histograms p and q is defined as:

$$\mathcal{M}_{p,q} = \frac{\sum_{i,j} \min(H^p(i,j), H^q(i,j))}{\sum_{i,j} H^p(i,j)}. \qquad (1)$$

Those patches whose score is above a threshold (typically 0.9) are skin-colored and assumed to belong to the hand.

Once each image is segmented, a list of patches in the image which match the stored histogram is created. Each patch has associated with it a center and an area. Figure 2 shows an example segmentation given the image in Figure 1. Each small square represents one patch. As can be seen, the color-based segmentation results in a clean separation of the hand from the background.

2.1.3 Extracting 3D Position

After the segmentation process, it is possible to extract the 3D location and 2D orientation of the hand. The center of mass of the segmented patches is used as the 2D position of the hand: $C_x = \sum_i p_{ix}/N$ and $C_y = \sum_i p_{iy}/N$ where N is the total number of patches that were considered part of the hand, and p_{ix} and p_{iy} denote the x and y coordinate of the center of the i'th patch.

The planar rotation (rotation about the camera axis) is estimated by fitting an ellipse to the segmented patches and computing the angle of its principal axis. This can be done by computing the second order moments of the patches:

$$m_{20} = \sum (C_x - p_{ix})^2 \qquad (2)$$

$$m_{02} = \sum (C_y - p_{iy})^2 \qquad (3)$$

$$m_{11} = \sum (C_y - p_{iy})(C_x - p_{ix}) \qquad (4)$$

Then the orientation of the principal axis is given by:

$$\theta = \frac{1}{2} \tan^{-1}\left(\frac{m_{20} - m_{02}}{2m_{11}}\right) \qquad (5)$$

To smooth out noise we compute a moving average for each of the above parameters. The small filled square and the oblique line in Figure 2 depict the computed center of mass and orientation, respectively, of the hand in Figure 1.

Since we are only using one camera, it is difficult to obtain accurate estimates of the absolute distance of the hand to the camera. However, as an interface, it is sufficient to compute the relative depth (i.e. the depth relative to some standard location). With this information it is possible to tell whether the target is coming closer or moving further away. One way

to compute this is to simply count the total area of the segmented patches. As the target moves closer it will occupy a larger section of the image and the total patch area will increase. However this is susceptable to interference from transient noise patches. We obtain more robust estimates by weighting the area of the patches by a Gaussian placed on the center of mass:

$$A = A_P \sum_i \exp\left(\frac{-(C_x - p_{ix})^2 - (C_y - p_{iy})^2}{S}\right) \quad (6)$$

Here A_P is the area that each patch is responsible for and S is a scaling constant.

Computing relative distance in this way requires a simple calibration step. On the first image the hand is assumed to be at some canonical position. The initial area, A_0, is computed for the initial frame and stored. Then on subsequent frames, depth estimates are obtained by computing the instantaneous area and comparing it with A_0: $C_z = \frac{A_0}{A}$. This quantity will be 1 if the target is exactly at the same position as in the calibration frame. The value will get smaller as the target approaches the camera, and larger as it moves away. Thus it provides an estimate of the relative distance of the target from the camera.

2.2 Control Strategies

Although the above computations can be carried out reasonably efficiently, in order to obtain real-time performance on our system (a Sun Sparcstation), two resource allocation techniques were used. These include a dynamic search window, and a technique for adaptive subsampling. Without these techniques the tracking algorithm can achieve a maximum frame rate of about $8 - 10$ frames/second. The next few sections describe these aspects of the system.

2.2.1 Modifying the Search Region

A dynamic search region is implemented to ignore irrelevant regions of the image. The system maintains a rectangular tracking window around segmented patches. For each subsequent frame only image patches within this window are searched. At each iteration, given the current segmentation, the boundaries for the search region for the next frame are computed as follows:

$x_{min} = \min_i(p_{ix}) - b$ and $x_{max} = \max_i(p_{ix}) - b$
$y_{min} = \min_i(p_{iy}) - b$ and $y_{max} = \max_i(p_{iy}) - b$

Thus at the next frame, only the image pixels in the rectangle defined by (x_{min}, y_{min}) and (x_{max}, y_{max}) are searched. The constant b (we use a value of 40 pixels) ensures that the window is slightly larger than the actual target boundaries.

2.2.2 Adaptive Subsampling

It is not necessary to check every pixel within the search window. Indeed it is too time consuming to do so. In order to speed things up, the system subsamples the image. That is, once an image patch at location (x, y) is checked, the next patch is chosen starting at location $(x + s, y)$. (If the end of the scan line has been reached, then the patch starting at $(0, y + s)$ is checked.) The issue of determining the best subsampling constant, s, is important but non-trivial. Subsampling affects both the accuracy and the speed of the tracking. If s is too large than the position estimates discussed in the last section will be too noisy. If s is too small then too much time will be taken up processing each image.

Rather than use predetermined s, the system uses an adaptive technique to automatically select the best subsampling. The key is to allow the client application which is using the tracker to specify a goal frame rate. The segmentation module then continuously monitors its speed. If the segmentation time is faster than desired, then s is decreased. Conversely, if the time is slower than desired then s is increased to give greater speed.

2.2.3 Balancing Accuracy and Speed at Varying Depths

The above two techniques also solve a common problem concerned with the three-way interaction between accuracy, speed, and depth. In particular, when the target is close to the camera, the resulting image of the hand is large. In this case, the search window will be large, requiring more processing time per frame. s will be automatically increased, maintaining system throughput. When the target is far from the camera, the resulting image is small and so a smaller value of s is required in order to maintain accuracy. In this case the search window will be correspondingly small, requiring less processing per frame, and so the subsampling will be automatically decreased. The end result is that the system maintains relatively constant accuracy and speed at different depths. (It is easy to see that no fixed value of s can achieve this result.)

2.3 Using the Hand Tracker as a User Interface

The tracking system has been used to successfully navigate in virtual environments. Figure 3 shows the setup. The camera images the hand from above. Hand movements are transmitted to a real time VR simulator so that by moving his or her hand the user is able to translate in all six directions (up, down, left, right, forward, and backward) and rotate the view clockwise or counterclockwise. A 3D rendering of the hand provides positional feedback to the user for navigation. By including objects in the world that perform actions when touched the user is able to actually manipulate both the internal and external world. For example, we have implemented a CD player control panel, which when touched, plays back music using a CD ROM.

3 A 19 Degree of Freedom Hand Tracker

So far we have described a system that is able to track the 3D position and planar orientation of the hand. It is also of interest to recover the complete configuration of the hand. From a computer vision point of view, this is a very difficult task as the hand

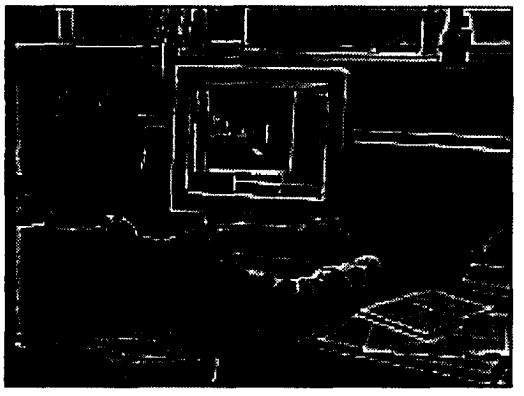

Figure 3: The view of the hand tracking setup.

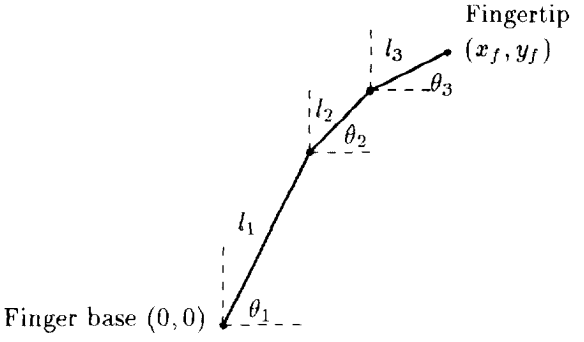

Figure 4: A model of the joint angles in a finger.

contains about 20 degrees of freedom (each finger contains four degrees of freedom: two joints with one degree of freedom and one joint, the base, with two). This section describes one approach for recovering the required information from the image.

3.1 Recovering Joint Angles

Figure 4 shows a schematized side view of a finger and its joints (the fourth degree of freedom, moving a finger from side to side is not considered here). Given the joint angles θ_i and finger segment lengths l_i, we can easily compute the location of the fingertip:

$$x_f = l_1 \cos(\theta_1) + l_2 \cos(\theta_2) + l_3 \cos(\theta_3) \quad (7)$$

$$y_f = l_1 \sin(\theta_1) + l_2 \sin(\theta_2) + l_3 \sin(\theta_3) \quad (8)$$

This is complicated by the fact that not every point in joint space is realizable with our fingers. For example, although there are three joints shown, the finger can only cover a two dimensional space. Even in 2D space, not every point is reachable (e.g. we cannot bend our finger backward). It turns out that if location of the fingertip is known, the set of possible joint angles is actually very constrained. This implies that, given the location of the fingertips, it is theoretically possible to approximately compute the actual joint angles. Unfortunately it is difficult to do this directly: there does not seem to be a closed form analytic expression for the inverses to equations (7) and (8) that can easily incorporate the joint angle constraints.

The approach we have taken has been to learn the inverse mapping. The idea is that the forward model is easy to compute. Given the 3 angles and knowledge of the segment lengths, we can compute the location of the fingertip from equations (7) and (8). So it is possible to construct a training set with pairs of vectors consisting of joint angles and corresponding fingertip locations. Using this training set, we can simply train a function approximator to compute the inverse map.

This turns out to be quite successful. For our purposes the best results were obtained using nearest neighbor approximation. That is given a fingertip location, locate the closest fingertip in the training set and select the corresponding joint angles. Using kd-trees one can implement nearest neighbor very efficiently [2]. Average retrieval time is $O(\log n)$ where n is the number of vectors in the training set.

3.2 Fingertip Detection

The joint angle recovery has been incorporated within the hand tracking system described earlier. In order to do this a "fingertip detector" had to be implemented. For efficiency reasons we decided on fitting a very coarse model of the hand to the segmented image patches. The palm is represented as a circle; the fingers are represented by lines emanating from the center of the wrist. The lines are allowed to rotate around the wrist center. To fit such a model we need to locate the radius of the palm and determine the angles and lengths of each finger.

In order to locate the circle representing the palm, the current system assumes that the center of the palm is at the center of mass. This is not completely accurate but is a reasonable assumption as long as the arm does not also appear in the segmented patches (i.e. the person should wear a full sleeved shirt). In order to find the palm the system computes the largest circle such that the area of all patches within the circle is roughly the same as the actual area of the circle.

Once the palm is located, the fingertips need to be detected. Every patch that is outside the circle could be part of a finger. In a human hand the finger bones emanate from a central point on the wrist. Thus the angles of the fingers (especially the thumb) are best measured from the wrist center. This point is obtained by projecting the center of mass backwards along the orientation of the hand to the edge of the circle.

In order to locate the actual angle of each finger we have used a Hough transform based approach. The set of possible angles are discretized into a number of bins. Each patch outside the circle has associated with it an angle to the wrist center and votes for this angle. Within each bin the system also keeps track of the patch with maximal distance that had voted for it. After the voting process the five best peaks in the histogram are selected and the maximal patch is chosen as the fingertip location.

3.3 Real Time Finger Modeling

The fingertip detector and joint angle recovery modules have been combined to obtain a real time system for approximating the user's finger movements. The system works, as before, by using the first frame as a calibration frame. In this frame the users hand is assumed to be flat with all fingers extended. The depth of the hand and the maximal length of each finger are stored. Then on each subsequent frame, each fingertip is located. Treating each finger independently, the system uses the joint angle module to output the joint angles for each finger to a visualizer. Each finger is only allowed three degrees of freedom (fingers are not allowed to wiggle from side to side). This results in 15 degrees of freedom for the fingers. Combined with the position and angle estimates leads to a total of 19 degrees of freedom. The combined system can run at a rate of 10 frames per second on a SparcStation 10. Although it often works well, this system unfortunately is not as stable as the hand tracker. This is due in part to failures in detecting the fingertip when the user's hand is not approximately parallel to the camera plane. When the fingertips are detected correctly then the visualizer shows a reasonable interpretation of the user's fingers moving around. Further research needs to be conducted on the feature detection stage.

4 Discussion

This paper has described initial steps towards building a complete hand recognition system based on computer vision. There are still several drawbacks of the systems described above. First, only rotations parallel to the camera plane are recovered. Rotating the hand around the other two axes can confuse the system. Second, although the system is quite insensitive to the image background, only one skin colored object may be present in the image at any one time. This means that the system cannot yet handle two hands in the image. Finally, the system for recovering joint angles occasionally has problems detecting the fingertip, mainly due to the limitations of the hand model used.

Apart from these disadvantages, the systems are quite stable and robust, particularly the position and orientation tracking component. There is typically very little noise in the position estimates. The approach based on fitting a set of patches to a model has proven to be very robust to noise in the segmentation process. Finally, the speed of the system provides the user with rapid feedback, an important factor in interactive systems.

How far away are we from full scale $3D$ hand tracking? There are a number of outstanding problems that must be solved. There are still no general algorithms for dealing with self-occlusion, for modeling non-rigid objects, and for determining the pose of non-rigid objects. Adaptive techniques based on density estimation[1] and on estimating the surface that data points lie on[3] appear promising. Although the task is formidable, the results outlined here provide some hope that vision based algorithms can eventually lead to non-obstrusive and natural $3D$ user interfaces.

Acknowledgements

Portions of this research were done at Siemens Central Research, Munich, Germany. I thank Daniel Goryn, Christoph Maggioni, Rolf Schuster, Brigitte Wirtz and Volker Tresp for helpful comments and suggestions.

References

[1] S. Ahmad and V. Tresp. Some solutions to the missing feature problem in vision. In S.J. Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 393–400. Morgan Kaufmann Publishers, 1993.

[2] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] Christoph Bregler and Stephen M. Omohundro. Surface learning with applications to lipreading. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*. pages 43–50. Morgan Kaufmann Publishers, 1994.

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

[5] J. Segen. Model learning and recognition of non-rigid objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Diego, CA, 1989.

[6] J.M. Stroud. The fine structure of psychological time. In H. Quastler, editor, *Information theory in psychology.* Free Press, Glencoe, Ill., 1956.

[7] M.J. Swain and D.H. Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.

[8] B. Wirtz and C. Maggioni. Imageglove: A novel way to control virtual environments. In *Proceedings of Virtual Reality Systems '93*, New York, 1993.

[9] T.G. Zimmermann, J. Lanier, C. Blanchard, and S. Bryson. A hand gesture interface device. In *Proceedings ACM CHI+GI Conference: Human Factors in Computing Systems and Graphics Interface*, pages 189–192, 1987.