

# **Real-time Hand Tracking and Finger Tracking for Interaction**

## ***CSC2503F Project Report***

**By: Shahzad Malik (smalik@cs.toronto.edu)**

***December 18, 2003***

### **1. Introduction**

Vision-based hand tracking is an important problem in the field of human-computer interaction, since hand motions and gestures could potentially be used to interact with computers in more natural ways. A number of solutions have been proposed in the current literature, but the problem is still far from being solved since the hand exhibits significant amounts of articulation and self-occlusion that cause difficulties with existing algorithms. To further exasperate these problems, interactive applications require that the hand tracking perform in real-time.

This project presents the implementation and analysis of a real-time stereo vision hand tracking system that can be used for interaction purposes. The system uses two low-cost web cameras mounted above the work area and facing downward. In real-time, the system can track the 3D position and 2D orientation of the thumb and index finger of each hand without the use of special markers or gloves, resulting in up to 8 degrees of freedom for each hand.

### **2. Related Work**

Hand tracking is an active area of research in the vision community, mainly for the purposes of sign language recognition and human-computer interaction. One of the original tracking systems to focus on articulated hand motion was presented in [Rehg93]. In their system, a 27 degree-of-freedom hand could be tracked at 10Hz by extracting point and line features from grayscale images. However, it has difficulty tracking in the presence of occlusions and complicated backgrounds, and it requires a manual initialization step before tracking can begin.

From an interaction perspective, most of the hand tracking work to date has focused on 2D interfaces. In [Zhang01], a finger was tracked across a planar region using low-cost web cameras in order to manipulate a traditional graphical interface without a mouse or keyboard. Fingertip detection was accomplished by fitting a conic to rounded features, and local tracking of the tip was performed using Kalman filtering.

Similarly, in [Sato00] infrared cameras were used to segment skin regions from background pixels in order to track two hands for interaction on a 2D tabletop display. Their method then used a template matching approach in order to recognize a small set of gestures that could be interpreted as interface commands. However, no precise fingertip position information was obtained using their technique.

### 3. System Overview

This section describes the implementation details of the hand tracking system, which is primarily based on the single hand tracker presented in [Segen99]. The system can extract the 3D position and 2D orientation of the index finger for each hand, and when present the pose of the thumb as well. In interactive applications a single pointing gesture could then be used for selection operations while both the thumb and index finger could be used together for pinching gestures in order to grasp and manipulate virtual objects.

#### 3.1 Background subtraction

The first phase of the tracking system involves separating potential hand pixels from non-hand pixels.

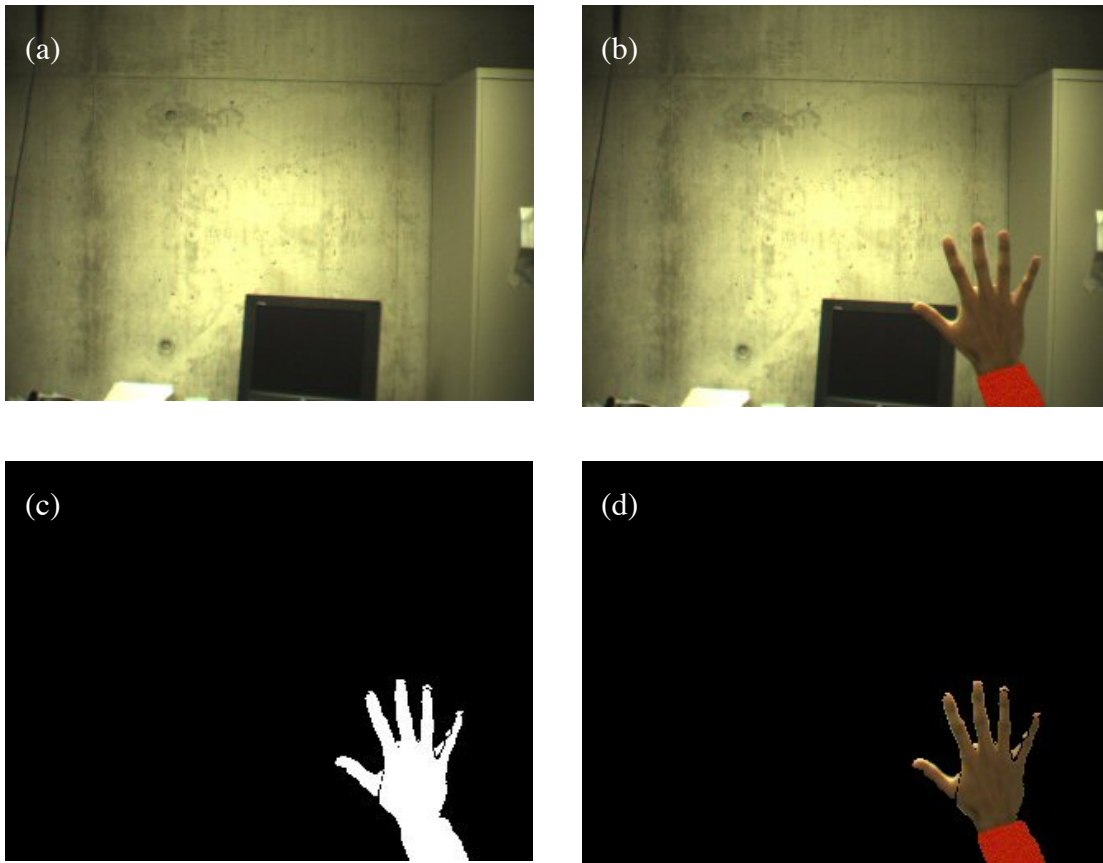
Before segmentation occurs, we first convolve all captured images with a 5x5 Gaussian filter and then scale this filtered image by one half in each dimension in order to reduce noisy pixel data. All subsequent image processing operations are then performed on this scaled and filtered image.

Since the stereo cameras are mounted above a non-moving workspace, a simple background subtraction scheme is used to segment any potential foreground hand information from the non-changing background scene. At system startup, a pair of background images  $I_{B,L}$  and  $I_{B,R}$  are captured to represent the static workspace from each camera view. Subsequent frames then use the appropriate background image to segment out moving foreground data. In other words, for each pixel in frame  $i$ , we compute the foreground mask image  $I_F$  (for each camera) as:

$$I_F = \begin{cases} 255 & \text{if } |I_i - I_B| > \sigma_B \\ 0 & \text{otherwise} \end{cases}$$

where  $\sigma_B$  is a fixed threshold to differentiate foreground data from background data. Note that background subtraction is performed in RGB colour space with 8-bits per colour channel. The resulting  $I_F$  is a binary image with a single 8-bit channel. After some experimentation, a  $\sigma_B$  value of 8 was found to provide good results.

Figure 1 shows the result of background subtraction on an image containing a hand.



**FIGURE 1 – Background subtraction: (a) Background image; (b) Captured image; (c) Foreground mask; (c) Foreground image**

### ***3.2 Skin Segmentation***

Although the background subtraction scheme described above works fairly well in segmenting foreground data from the non-changing background, it will still allow objects such as shirt sleeves, coffee mugs, or other desktop items that are placed into the workspace to be detected as potential hands. In order to deal with such situations and add some more flexibility to the system, a skin pixel detector has been implemented to further filter the foreground data.

As a pre-processing step, for each camera a small number of snapshots are taken of various hands with a range of different skin-tones and poses. Then using an image editing program each of the captured images is manually segmented into a binary mask where white pixels represent skin areas and black pixels represent non-skin areas. This set of captured images and associated skin masks is then used as the training set for a histogram-based skin classifier as described in [Jones99].

Using a bin size of 32 for each colour channel, each of the RGB pixels in the training set are assigned to either the 3D skin histogram  $H_s$  or the non-skin histogram  $H_n$ . Given

these histograms we can then compute the probability that a given RGB colour belongs to the skin and non-skin classes as follows:

$$P(rgb | skin) = \frac{s[rgb]}{T_s}$$

$$P(rgb | \neg skin) = \frac{n[rgb]}{T_n}$$

where  $s[rgb]$  is the pixel count in bin  $rgb$  of  $H_s$ ,  $n[rgb]$  is the pixel count in bin  $rgb$  of  $H_n$ , and  $T_s$  and  $T_n$  represent the total counts contained in  $H_s$  and  $H_n$  respectively.

Therefore, at run-time, we can determine the probability that any given  $rgb$  pixel is skin or non-skin using Bayes rule:

$$P(skin | rgb) = \frac{P(rgb | skin)P(skin)}{P(rgb | skin)P(skin) + P(rgb | \neg skin)P(\neg skin)}$$

where  $P(skin)$  and  $P(\neg skin)$  are our prior probabilities for skin and non-skin respectively.

Since  $P(skin) + P(\neg skin) = 1$ , we use  $P(skin) = \frac{T_s}{T_s + T_n}$  and  $P(\neg skin) = 1 - P(skin)$ .

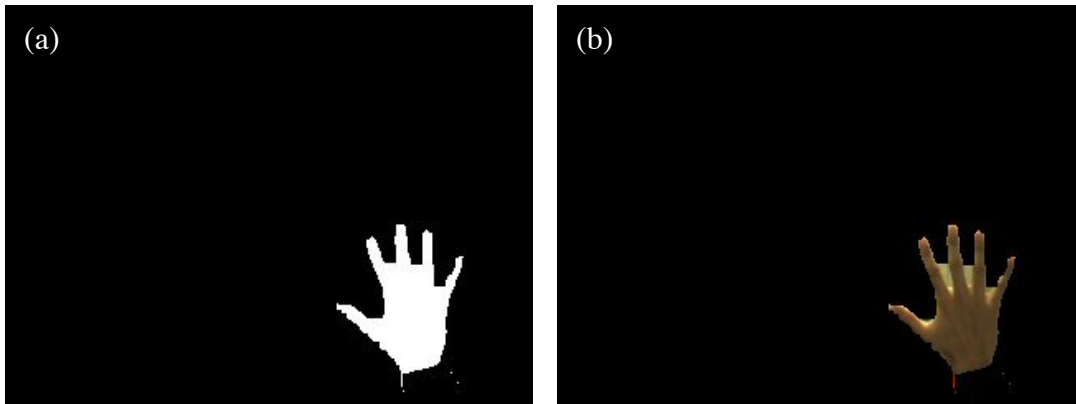
Thus we can further threshold our background subtracted image with this skin classifier to only keep pixels with a high skin probability:

$$P(skin | rgb) \geq \sigma_s$$

where  $\sigma_s \in [0..1]$  is our threshold value. After some experimentation, a value of 0.6 was found to provide good results for  $\sigma_s$ .

The result of our skin classifier is a new binary skin mask image  $I_S$ . Since not all skin pixels will be categorized correctly at all times, we perform a morphological *closing* operation on  $I_S$  in order to remove small noisy holes in the skin pixel areas.

Figure 2 shows the result of skin detection using the background subtracted image in Figure 1d.



**FIGURE 2 – Skin detection: (a) Skin mask; (b) Foreground skin image**

### 3.3 Region Extraction

Now that the skin regions have been detected, we must determine which regions correspond to the left and right hands. It is possible that small noisy regions will still be present after background subtraction and skin segmentation, but we assume that the regions corresponding to the hands will be the largest.

Thus we first extract the contours of all the detected skin regions in  $I_S$  using binary image processing operations and connected component analysis. For each region  $i$  we thus get a set of counter-clockwise perimeter coordinates  $C_i(j) = \{ (x_j, y_j) \}$  that trace the outline of each region.

Let  $N_i = |C_i|$  represent the total number of perimeter coordinates in the contour  $i$ . We then choose the two largest contours  $A$  and  $B$  to represent the hand contours, using  $N_i$  as a measure of contour size. Additionally, in order to avoid processing extremely small contours,  $N_i$  must be above some threshold  $\sigma_N$  in order for the contour  $i$  to be considered valid ( $\sigma_N = 50$  in the current implementation).

We then compute the mean of each of these two largest contours by averaging the perimeter coordinates in  $C_A$  and  $C_B$  respectively. This is followed by a simple heuristic approach to differentiate between the left and right hands by stating that the contour with the smaller mean  $x$  coordinate is the left hand, and the contour with the larger mean  $x$  coordinate is the right hand (assuming the image  $x$  coordinates increase from left to right). In the case where only a single large contour has been detected, the system sets it to be the right hand under the assumption that the user is right-handed and will be using their dominant hand for one-handed operations (this default can be changed for left-handed users).

Figure 3 shows the result of the contour extraction on the image from Figure 2.



**FIGURE 3 – Hand contours**

### 3.4 Feature Extraction

Let  $C_L$  represent the contour for the left hand, and  $C_R$  represent the contour for the right hand as computed above. In order to find the fingertips for the thumb and index finger for each hand, we attempt to find pixels that represent peaks along the contour perimeters.

At each pixel  $j$  in a hand contour  $i$ , we compute the k-curvature which is the angle between the two vectors  $[C_i(j), C_i(j - k)]$  and  $[C_i(j), C_i(j + k)]$ , where  $k$  is a constant (currently set to 16). The k-curvature can be computed quite easily using a dot product operation between the vectors. The idea here is that contour points with a k-curvature close to 0 will represent potential peaks or valleys along the perimeter. We currently use a degree threshold  $\theta_k = 30$  for the k-curvature such that only points below this angle will be considered further.

In order to classify the points as either peaks or valleys, we convert the vectors into 3D vectors lying in the  $xy$ -plane and then compute the cross product. If the sign of the  $z$  component of the cross product is positive then we label the point as a peak, while a negative cross product results in a valley label.

Finally, non-maximal suppression is then used to find the strongest peaks and valleys along the perimeter, since we can expect that a sequential set of peaks and valleys will be detected in the neighbourhood of the strongest locations.

Figure 4 shows the result of peak and valley detection on the image in Figure 3. Note that not all valleys were detected, largely as a result of the morphological closing operation that was performed during skin segmentation, but this will not be a problem for gesture recognition as described in the next section.



**FIGURE 4 – Peak and valley detection**

### 3.5 Point and Pinch Gesture Recognition

After the feature extraction phase, we have 2D positions for the peaks and valleys along the contours of the hand regions. For interaction purposes we can now recognize pointing gestures and pinching gestures as follows:

*Pointing Gesture: 1 peak*

*Pinching Gesture: 2 peaks*

For the pointing gesture, we assume that the single peak represents the index finger and no other finger is present. For the pinching gesture however, we must differentiate between the thumb and index finger peaks. Since the contour perimeter is given in a counter-clockwise order, we can use a simple heuristic to label the peaks as thumb or index finger. Define  $\text{cnorm}(x)$  as:

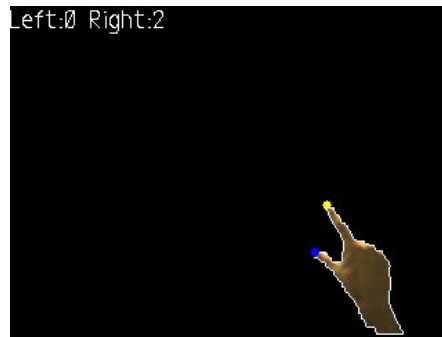
$$\text{cnorm}(x) = \begin{cases} x + N : x < 0 \\ x - N : (x + 1) > N \\ x : \text{otherwise} \end{cases}$$

where  $N$  represents the number of points along the contour perimeter. Let  $P$  and  $Q$  represent two peaks located at positions  $p$  and  $q$  in the counter-clockwise perimeter respectively. Therefore  $\text{cnorm}(p - q)$  gives us the distance between  $P$  and  $Q$  when traveling in the counter-clockwise direction along the contour from  $Q$  to  $P$ .

Given the left hand contour with two peaks, we know that the distance from the thumb to the index finger will be shorter than the distance from the index finger to the thumb (in the counter-clockwise order). Therefore if  $\text{cnorm}(p - q) < N/2$ , then  $P$  is the index finger and  $Q$  is the thumb, otherwise  $Q$  is the index finger and  $P$  is the thumb.

Similarly, for the right hand contour with two peaks, we know that the distance from the index finger to the thumb should be less than the distance from the thumb to the index finger. Therefore if  $\text{cnorm}(p - q) < N/2$ , then  $Q$  is the index finger and  $P$  is the thumb, otherwise  $P$  is the index finger and  $Q$  is the thumb.

Using this technique, we can properly label the thumb and index finger in a rotation invariant manner. Figure 5 shows the result of pinch gesture recognition on an image.



**FIGURE 5 – Gesture recognition results (a yellow dot represents the tip of the index finger, while a blue dot represents the tip of a thumb)**

### 3.6 2D Pose Estimation

The previous section described how to determine the position of the index finger and thumb. For interaction purposes, it would also be useful to determine the orientation of the fingers.

Let  $P(i) = (x_i, y_i)$  represent the  $i$ -th point along a hand's contour. Then  $P(\text{cnorm}(i + k))$  denotes a point that is  $k$  points to the left of  $P(i)$  along the perimeter, while  $P(\text{cnorm}(i - k))$  represents a point to the right. Let  $i_{ft}$  represent the index of a finger tip (thumb or index finger) that we wish to determine the orientation of. A midpoint  $Q(k)$  can then be computed as:

$$Q(k) = \frac{P(\text{cnorm}(i_{ft} + k)) + P(\text{cnorm}(i_{ft} - k))}{2}$$

Therefore we compute  $Q(k)$  for  $k_{min} < k < k_{max}$ , which gives us a set of midpoints representing the support for the axis of the finger. A linear least squares line fitting is then performed using these midpoints, as shown in Figure 6.

Let  $ax + by + c = 0$  represent the equation for the axis of the finger, where  $a^2 + b^2 = 1$ . We would like to minimize the sum of the squared residuals:

$$\min \sum_i (ax_i + by_i + c)^2$$

This can be done as follows:

1) Compute the centroid  $(\bar{x}, \bar{y})$  of the point set:

$$\begin{aligned}\bar{x} &= \frac{1}{n} \sum x_i \\ \bar{y} &= \frac{1}{n} \sum y_i\end{aligned}$$

2) Change coordinates so that new centroid is (0,0):

$$\begin{aligned}x'_i &= x_i - \bar{x} \\ y'_i &= y_i - \bar{y}\end{aligned}$$

3) Solve for  $(a, b)$  by minimizing the following quadratic form:

$$e(a, b) = \sum_i (ax'_i + by'_i)^2$$

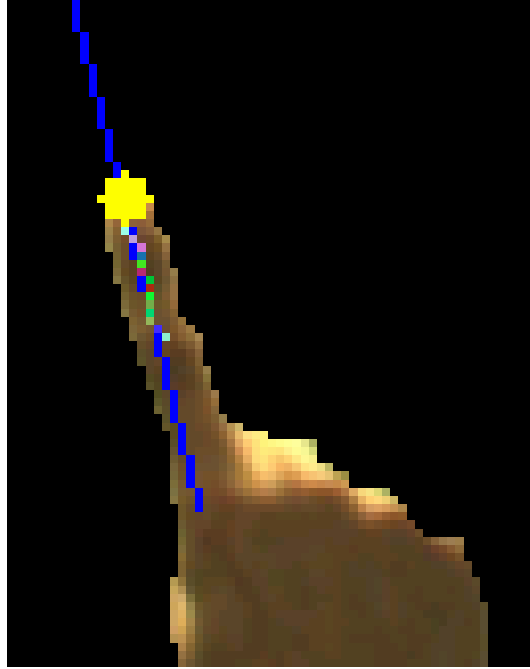
This can be solved by computing the SVD of  $M$ , where

$$M = \begin{bmatrix} \vdots & \vdots \\ x'_i & y'_i \\ \vdots & \vdots \end{bmatrix}$$

Therefore  $n = (a, b)^T$  is the second column of the  $V$  matrix from the SVD. Then  $c$  can be solved by computing  $c = (\bar{x}, \bar{y}) \cdot n$ .



Finally from  $n$  we can compute the  $\theta$  orientation of the finger, resulting in three parameters  $(x,y,\theta)$  for each of the detected finger tips in the stereo images. It is worthwhile to mention that while a robust M-estimation technique could be used to estimate the line, this may not be required since we are using midpoints of contour points along the finger. Thus outliers will usually only occur if the  $k_{max}$  value for  $Q(k)$  is too large, resulting in midpoints from non-finger contour positions. Therefore by controlling  $k_{max}$  we can reduce most outliers automatically, but this also reduces the number of data points for our line support, so a tradeoff has to be made here.



**FIGURE 6 – Least squares line fitting using midpoints**

### 3.7 3D Pose Estimation

Before we can determine the 3D position of the fingertips, the intrinsic and extrinsic camera parameters must be computed. As a pre-processing step, a simple black and white planar checkerboard pattern is captured at four different poses using the stereo cameras. These images are then passed to the Intel OpenCV Calibration Toolbox in MATLAB where corresponding corner features are manually selected in each image. The calibration utility then outputs the intrinsic camera parameters  $K_L$  and  $K_R$  for each camera by optimizing across the entire calibration sequence, as well as the pose of the checkerboard in each image (the extrinsic parameters). As described in [Trucco99], for our stereo camera setup we must determine the rigid transformation  $R$  and  $T$  between the two cameras in order to triangulate the 3D position of image features. Using the left camera as the reference coordinate frame, the rigid transformation from the right camera coordinate frame to the left camera coordinate frame is computed as:

$$R = R_R R_L^T$$

$$T = T_L - R^T T_R$$

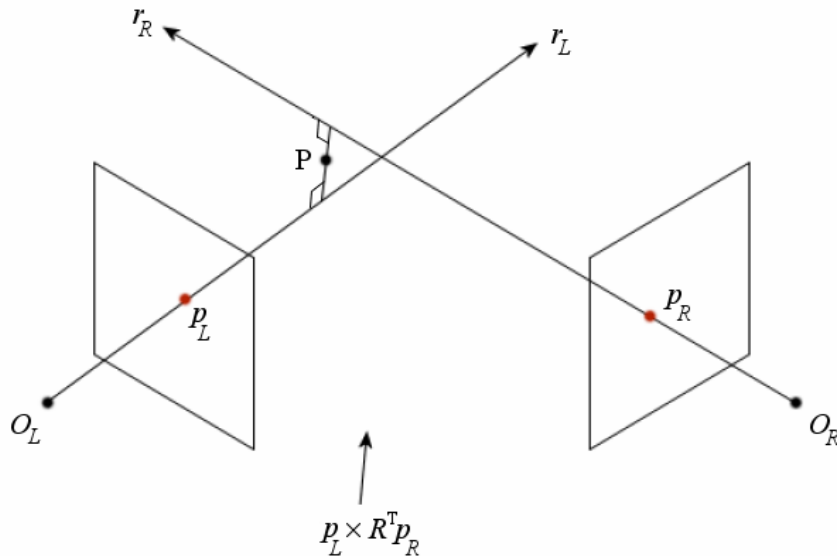
where  $R_L$  and  $T_L$  represent the extrinsic parameters of the left camera (rotation and translation of the checkerboard) from the first left frame of the calibration sequence, and similarly  $R_R$  and  $T_R$  represent the extrinsic parameters for the right camera using the first right frame of the camera calibration sequence. Although better results could be obtained by optimizing  $R$  and  $T$  across the entire calibration sequence, this simple method of using only the extrinsic data from the first frame of the sequence should be sufficient for our purposes.

With  $K_L$ ,  $K_R$ ,  $R$ , and  $T$ , we can now triangulate the 3D location of corresponding points  $p_L$  and  $p_R$  from the left and right images respectively using the technique described in [Trucco99]. The basic idea (depicted in Figure 7) is as follows:

- 1) Compute the 3D ray  $r_L$  that goes from the centre of projection  $O_L$  of the left camera and passes through  $p_L$ . Therefore  $r_L = ap_L$  ( $a \in \mathbb{R}$ ).
- 2) Compute the 3D ray  $r_R$  that goes from the centre of projection  $O_R$  of the right camera and passes through  $p_R$ , represented in the left camera reference frame using  $R$  and  $T$ . Therefore  $r_R = T + bR^T p_R$  ( $b \in \mathbb{R}$ ).
- 3) Compute the intersection point  $P$  of the two rays as the reconstructed 3D point. Since the rays may not truly intersect due to calibration and feature point inaccuracies, the 3D point  $P$  is computed as the midpoint of the smallest connecting line segment that is perpendicular to both rays. If we let  $a_0$  and  $b_0$  represent the endpoints of this line segment, then we can solve for  $a_0$ ,  $b_0$ , and  $c_0$  with the following linear system:

$$ap_L - bR^T p_R + c(p_L \times R^T p_R) = T$$

The resulting 3D point will thus be in the coordinate frame of the left camera (our chosen reference frame).



**FIGURE 7 – Triangulation with non-intersecting rays [Trucco99]**

## 4. Results

In this section we describe the accuracy and performance of the hand tracking software. The system was implemented in C++ under Microsoft Visual Studio, using the OpenCV and IPL libraries for image processing operations and OpenGL for display purposes. The system was tested on a Pentium 4 processor running at 2 GHz. The images were captured using a pair of DragonFly cameras with FireWire connections, providing us with 640x480 24-bit images and a capture rate of 30Hz. As mentioned earlier, the intrinsic and extrinsic camera parameters were computed using the OpenCV Calibration Toolbox in MATLAB, and the rigid transformation from the right camera frame to the left camera frame was also computed using MATLAB. Overall, the system can track the hands and fingertips at about 15Hz, which is quite good for interactive applications.

### *4.1 Peak and Valley Detection Performance*

Since the gesture recognition system relies on the location of fingertip peaks, it is worthwhile to first examine the peak and valley detection performance.

The following image shows the detection of all five fingertips of the hand. Peak detection works fairly well, but as can be seen the valley detection is somewhat sensitive to the separation of the fingertips. This is largely due to the morphological closing operation that is performed to fill in noisy skin pixels. While the current set of gestures do not rely on valley detection, it is worthwhile to consider improvements in this area if we wish to leverage the valley information in the future.

False negatives for peak detection tend to occur frequently for the thumb, largely as a result of our  $k$ -curvature constant. Since the thumb is shorter than most other fingers, it is sensitive to large values for this constant. However, decreasing the  $k$ -curvature would increase the false positive rate for the peak detector, so the current value of 16 provides reasonable overall performance. False negatives can also occur as a result of our choice of  $\theta_k$ , which defines the angle threshold for valid peaks and valleys. Increasing this value would allow more peaks and valleys to be detected, but would also increase the false positive rate.

The following image shows a thumb that has failed to be detected:



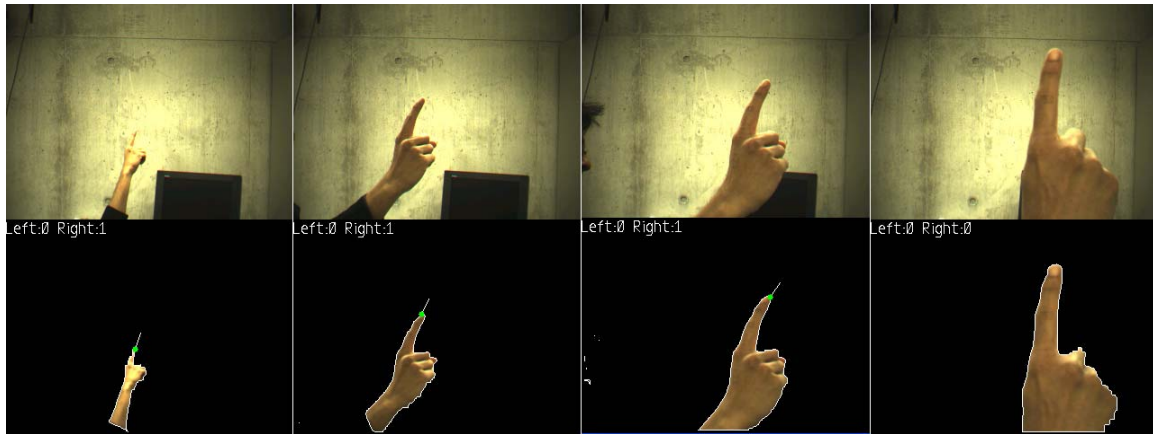
**FIGURE 8 – False negative for thumb detection**

False positives for both peaks and valleys can occur in areas where the skin segmentation has failed. Figure 9 shows an image where the skin classifier has failed to find all the skin pixels due to a hand appearing in a shadowed area. As a result, the peak detector has labeled a sharp contour point as a valid fingertip. A possible remedy to this situation would be a larger training set for our skin histogram in order to account for more skin tones and illumination conditions.



**FIGURE 9 – False positive peak detection**

Finally, although our peak and valley detection is rotation invariant it is still sensitive to changes in scale. The following image shows what happens when a hand is moved too close to the camera:

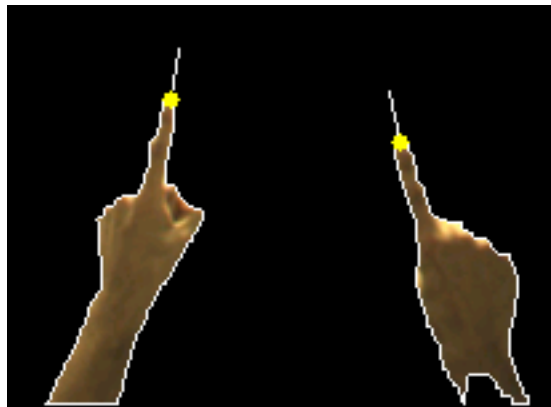


**FIGURE 10 – Scale sensitivity (the fingertip peak fails to be detected close to the camera)**

Again, this is a result of our  $k$ -curvature constant; for successful detection of hands close to the camera, it would be wise to increase this value. Nevertheless, a fixed  $k$ -curvature value provides a reasonable range in which the fingertips can still be detected successfully. An interesting future enhancement would be to dynamically modify the  $k$ -curvature value based on the distance of the hand from the camera, thereby allowing a form of scale invariance.

#### ***4.2 Gesture Recognition Performance***

Recognition of the pointing gesture works fairly well for both the left and right hands with the default threshold values. Figure 11 shows an example of the recognition of the pointing gesture with both hands present in the image.



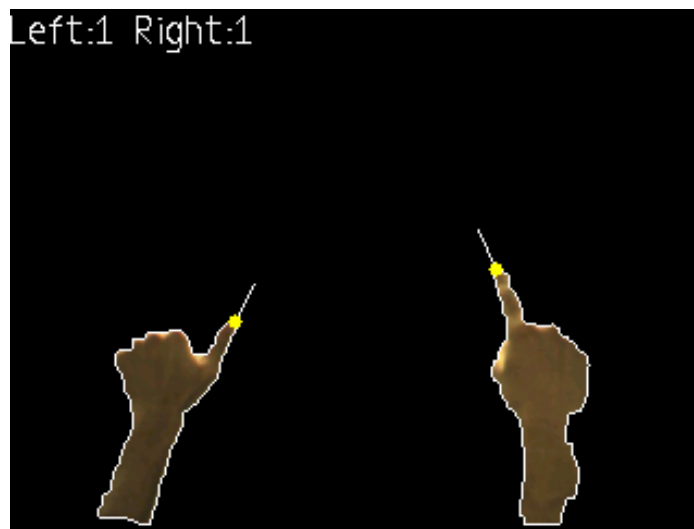
**FIGURE 11 – Successful recognition of the pointing gesture for each hand (yellow dot is tip of index finger)**

The pinching gesture is slightly more difficult to recognize since the shortness of the thumb is sensitive to our choice of the  $k$ -curvature constant. Nevertheless, in most cases the gesture is detected successfully, as depicted in the following image:



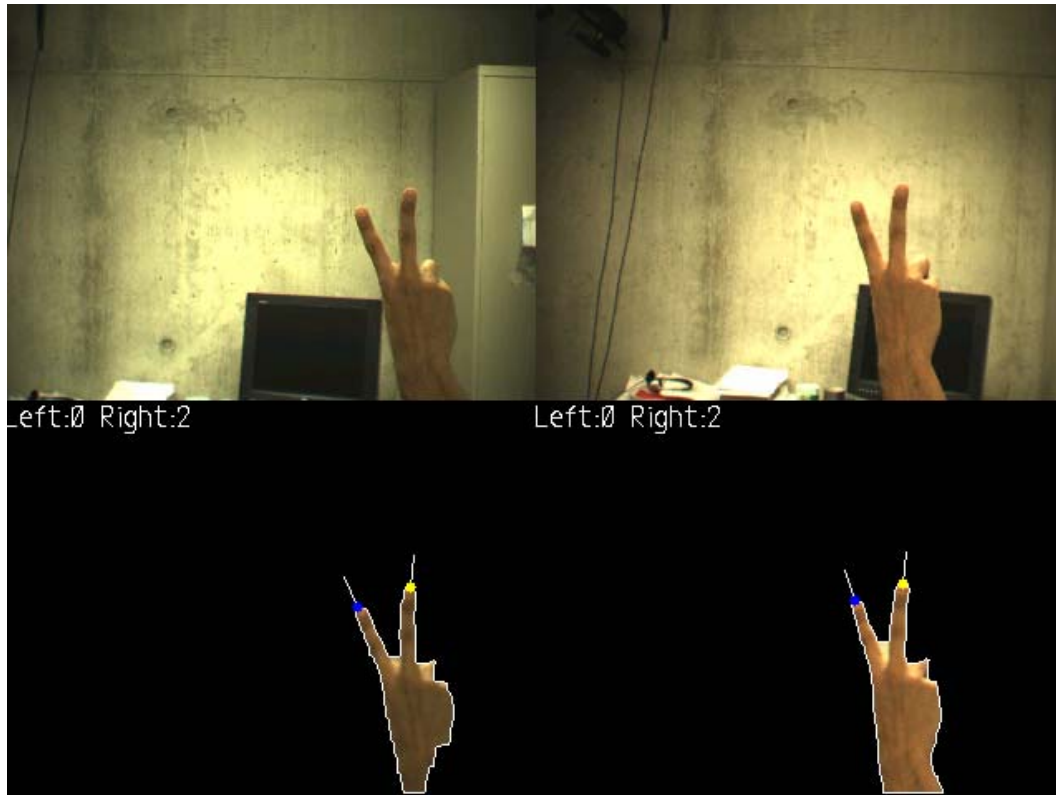
**FIGURE 12 – Successful recognition of the pinching gesture (yellow is tip of index finger, blue is thumb tip)**

Due to the simple heuristic approach for our gesture recognition, it is quite easy to fool the system. For example, showing any single finger will cause that fingertip to be labeled as the index finger. The following image shows such a situation:



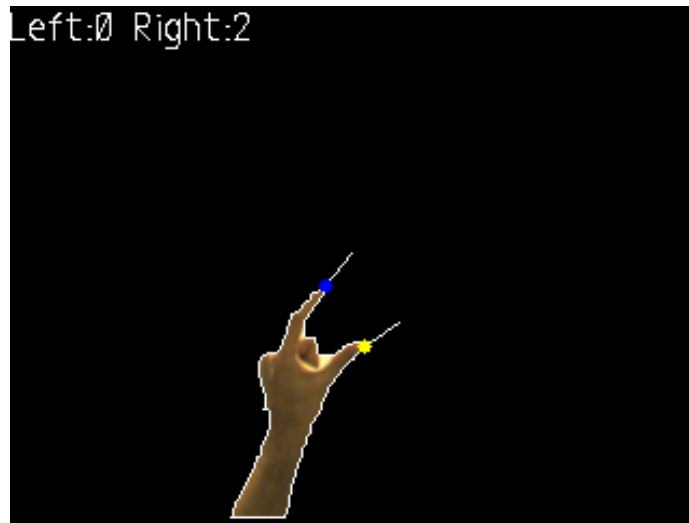
**FIGURE 13 – Any single fingertip is interpreted as the index finger. The right index finger is correct, but in the left hand the thumb is being interpreted as the index finger.**

Similarly, any two-finger gesture will result in one finger being labeled as the index finger and the other the thumb. The criteria used for the labeling will depend on the distance between the fingers along the hand contour. The following image shows such a situation:



**FIGURE 14 – An arbitrary two-finger gesture using the right hand is interpreted as a pinch gesture in a stereo image pair. Notice that one finger becomes the thumb (blue) and the other becomes the index finger (yellow).**

In most interaction tasks, the left and right hands will appear in the left and right sides of the image respectively. Thus our simple left hand/right hand detection heuristic that uses the mean contour position works well in most cases. Nevertheless, it is still possible to trick the system by simply crossing the hands over. Additionally, since the system currently assumes that a single hand in the scene should be interpreted as the right hand, then showing only the left hand results in incorrect labeling of the fingers as shown in the following image:

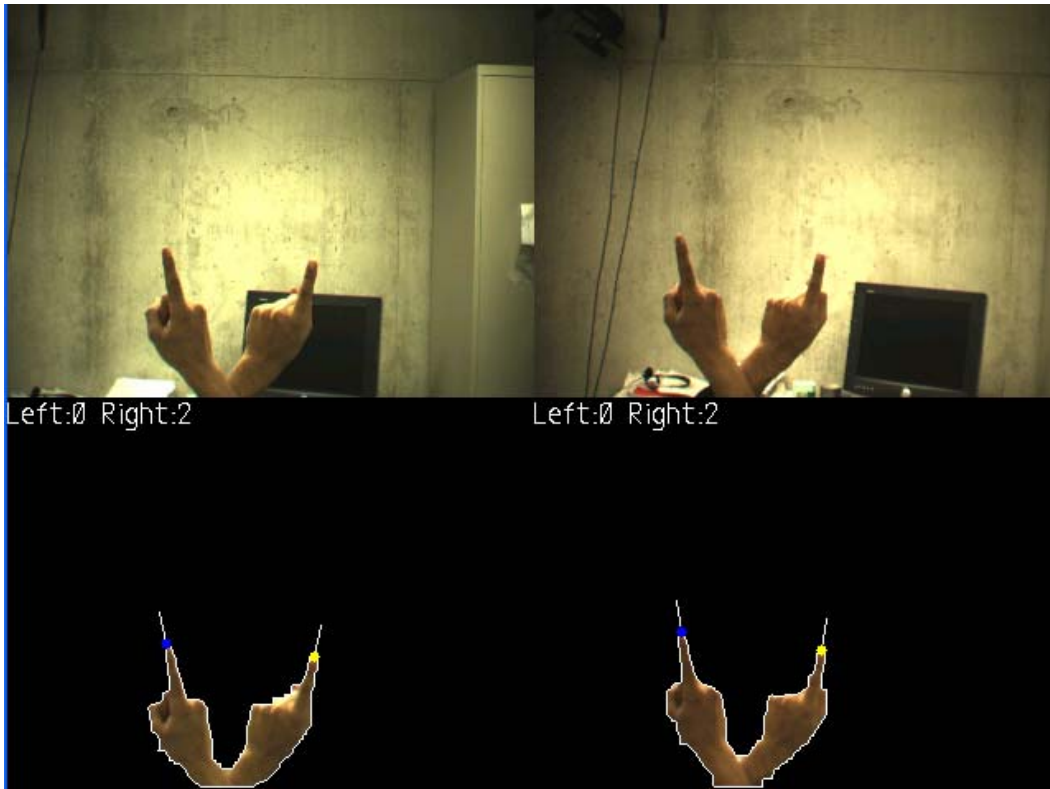


**FIGURE 15 – Left hand only is interpreted as a right hand (yellow is supposed to be the index finger, blue is supposed to be the thumb)**

For interaction purposes we assume that the cameras will be viewing the top of the hands. As a result, showing the palms of the hands instead of the tops will cause a similar misclassification of the fingertips as when we cross the hands over.

Another misclassification problem occurs when two hands appear close together in the captured images. This results in a single large region being segmented by the background subtraction and skin detection phases. Therefore the contour detector interprets the two hands as a single hand and thus the fingers are labeled as a right hand. The following image shows what happens when the left and right hands are crossed over:





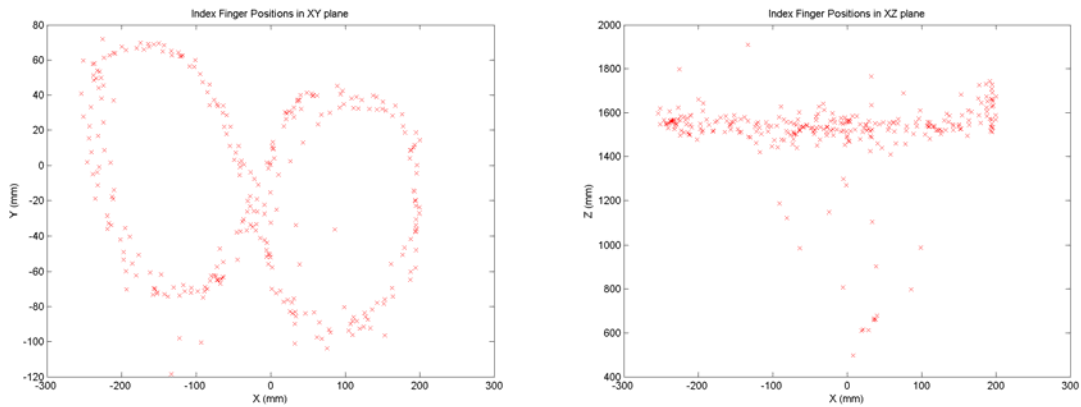
**FIGURE 16 – Two hands interpreted as a single right hand in the stereo images. Notice that one finger is labeled as the thumb (blue), the other the index finger (yellow).**

#### ***4.3 3D Position Measurement Accuracy***

Since it is difficult to measure any ground truth for the 3D position of fingertips, we instead measure the accuracy of the 3D reconstruction by moving the pointing gesture through a set of motions. For each pair of stereo frames, the reconstructed 3D fingertip position is dumped to a log file. We then plot these 3D positions in MATLAB and qualitatively assess our position accuracy.

The first motion consists of moving the finger tip in a figure-eight motion in the  $xy$ -plane so that we can analyze the accuracy of the  $x$  and  $y$  position reconstruction. Figure 17 shows the 3D plot of the figure-eight motion, and Figure 18 shows the corresponding 2D plots. As can be seen, the  $x$  and  $y$  positions have been reconstructed quite accurately, with some occasional noise due to incorrect correspondences as the hand is first brought into the scene. Interestingly, the figure-eight motion was drawn against a flat wall that was approximately 150 cm from the left camera mounted on a tripod. As expected, according to the 2D XZ plot the majority of the 3D points are also at the  $z = 150$  cm position.

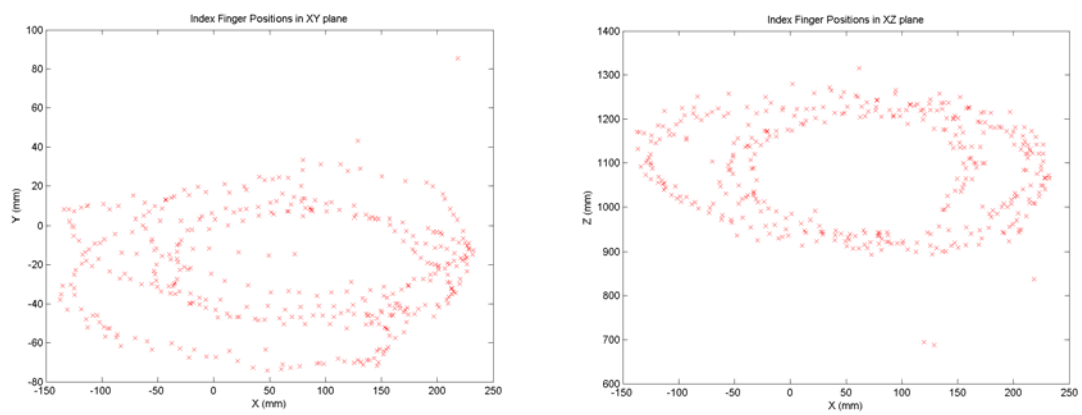
**FIGURE 17 – 3D plot of figure-eight motion in  $xy$ -plane (blue diamond represents camera origin)**



**FIGURE 18 – 2D plots of figure-eight motion in  $xy$ -plane**

The second motion involves moving the finger tip in a circular motion in the  $xz$ -plane, in order to see how accurately the  $z$  position (depth) is reconstructed. Figure 19 and Figure 20 plot the 3D and 2D motion data respectively for this circular finger motion. Again, the 3D positions have been reconstructed fairly well. Note that the spiral effect was caused by motion changes of the arm and not by some error in the reconstruction, since it is difficult to always trace out identical circles with the index finger in mid-air.

**FIGURE 19 – 3D plot of circular motion in xz-plane (blue diamond represents origin)**



**FIGURE 20 – 2D plots of circular finger motion in xz-plane**

## 5. Future Work

While the system works fairly well for the simple pointing and pinching gestures, there is still room for improvement. Currently the system assumes a static background, but it would be desirable to use this hand tracking system in an augmented reality setting where a user, wearing a head-mount display, could interact with virtual 3D objects in the real-world. In other words, the cameras would be attached to the head-mount display and viewpoint could thus be controlled by natural head motions, resulting in a changing background scene. If the skin pixel detector could be made more robust, it would be possible to completely discard the background subtraction phase and use the current system in such an augmented reality setting. However, a more sophisticated hand segmentation system would still be required in order to differentiate between other objects with skin-coloured pixels, such as faces.

Another improvement to the current system would be a more sophisticated gesture recognition system, possibly using template matching. Thus we could begin to differentiate between multiple single-finger gestures (currently the system assumes any single finger is the index finger). Additionally, a more sophisticated left/right hand detector would be an interesting area for future research to handle situations such as hands crossing over or views of the palms.

The current implementation performs a number of image processing operations on the entire image after every frame capture. While the speed of the tracker is quite good, we could possibly improve the tracking rate significantly by tracking local features of the hand instead (e.g. the fingertips). The downfall with this approach is that moving the hand too quickly could result in tracking failure, but we could always simply revert back to the current implementation to reinitialize the tracker.

In certain situations a fingertip feature could be detected in one stereo image but not the other due to image processing errors as described earlier. Thus another possible improvement would be to make use of epipolar geometry in order to redetect the fingertip in the second image. Additionally, epipolar geometry could also be used to make our stereo fingertip correspondences more robust by verifying that each detected feature in one image has its corresponding feature lying on or close to the epipolar line in the other image.

Finally, the current implementation only uses the 2D finger axis from either the left or right image as a measure of finger orientation. While this is sufficient for 2D interactions, it would be desirable to determine the 3D axis of the finger in order to detect finger orientations in the  $z$  (depth) direction as well. It turns out that this could be accomplished quite easily by computing two planes that pass through the finger from each camera. In other words, the first plane would pass through the centre of projection of the left camera and through the 2D image line for the finger in the left image. Similarly, the second plane would pass through the centre of projection of the right camera and through the 2D image line for the finger in the right image. The line of intersection between these two planes would then represent the axis of the finger in 3D,

providing us with two rotational degrees of freedom instead of the current one (the third axial rotation of the finger cannot be extracted directly using this method).

## 6. Conclusion

This project presented a vision-based hand tracking system that does not require any special markers or gloves and can operate in real-time on a commodity PC with low-cost cameras. Specifically, the system can track the tip positions of the thumb and index finger for each hand, assuming that a calibrated pair of cameras is viewing the hands from above with the palms facing downward. The motivation for this hand tracker was a desktop-based two-handed interaction system in which a user can select and manipulate 3D geometry in real-time using natural hand motions. The algorithmic details for the hand tracker were presented, followed by a discussion of the performance and accuracy of the system, as well as a discussion of how the system could be improved in the future.

## References

- [Jones99] M. Jones, J. Rehg. *Statistical color models with application to skin detection*. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1999. Vol. 1, pp. 274-280.
- [Rehg93] J. Rehg, T. Kanade. *DigitEyes: Vision-Based Human Hand-Tracking*. School of Computer Science Technical Report CMU-CS-93-220, Carnegie Mellon University, December 1993.
- [Sato00] Y. Sato, Y. Kobayashi, H. Koike. *Fast tracking of hands and fingertips in infrared images for augmented desk interface*. In Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition (FG), 2000. pp. 462-467.
- [Segen99] J. Segen, S. Kumar. *Shadow gestures: 3D hand pose estimation using a single camera*. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1999. Vol. 1, pp. 479-485.
- [Trucco98] E. Trucco, A. Verri. *Introductory Techniques for 3D Computer Vision*. Prentice-Hall, 1998.
- [Zhang01] Z. Zhang, Y. Wu, Y. Shan, S. Shafer. *Visual panel: Virtual mouse keyboard and 3d controller with an ordinary piece of paper*. In Proceedings of Perceptual User Interfaces, 2001.