Finger Tracking for Rapid Cropping Applications

Gunnar Sigurdsson, Andrew Wong
CS223B Winter Quarter 2008
Stanford University

1. Introduction

Remote collaboration is a hard reality for today's designer. In particular, remote brainstorming is nearly impossible despite advances in modern communication. While speaking and seeing each other over the Internet is now possible with modest bandwidth, the sharing and saving of dynamically drawn content is still severely lacking (see Figure 1). PDF standards, file transfer protocols, photo sharing applications all enable the sharing of ready made documents–but what about drawings made on the fly? This report investigates the use of a webcam to rapidly grab drawings on paper. In particular, we investigate the use of finger tracking as an interface for cropping areas of interest.

Figure 1. a) Brainstorm session at industrial design firm IDEO. Note the mass use of paper and pens to share information locally. b) State of the art teleconferencing with the Cisco Telepresence. Audio/visual clarity is maximized at the cost of content sharing.

Webcams now have high enough resolution to serve both as a video and a photo source. Our motivation for using it as an imaging source is speed. Scanners take on average 1-2 minutes to process a document. Digital cameras perform slightly better, taking about 30-60 seconds to get the content from the camera to the screen. Webcams stream video with latencies in the tens of milliseconds, and have the added bonus of easy interfacing with a computer as a sensor device.

In order to grab images quickly in a live environment, the interaction with the imaging system must be as intuitive and seamless as possible. We found circling ideas with the finger to be the gesture used in typical brainstorming environments, and thus a suitable candidate. Finger tracking enables the user to be free of input devices except for the common pen and paper. By employing this system along with an image capturing system, auto-contrasting, and finally image sharing application, we envision a teleconferencing tool that can support the needs of today's designers.

We broke down the problem of finger tracking into following steps:

- Hand Detection
- Finger tip extraction

For hand detection, we used an adaptive background subtraction method in the YUV color space with median filtering to generate a binary image of the hand. Finger tip extraction was then performed with shape filtering and an edge point classifier algorithms.

2. Prior work

Finger detection has been employed with a variety of different algorithms with mixed results. In our work, we attempted to grab the most useful algorithms from prior work, and tweak parameters to suit our case.

2.1. Hand detection

Segmenting the hand in prior work usually consists of some form of skin detection. In [2], a skin color histogram descriptor was constructed from thousands of skin images on the internet. This was then used to classify new images. We found this approach to be too general for our application, as most of the images we are detecting have only paper or hands.

To build a system insensitive to illumination changes, two different color spaces have been used in the past. [1] used something called Chrominance Euclidian Distance (CED), basically normalized red and blue values, as their metric, claiming it to be purely color-based and robust to shadows. [7] used the YUV color space for the same reason, claiming illumination can be ignored by operating only on the U and V-channels. Both techniques have their weaknesses under certain color conditions and we found that the

YUV to be most useful for our application. Unlike the paper, we found the CED method to be very sensitive to shadows.

Background subtraction methods have generally been mildly successful with some sort of adaptive approach. [1] updated the background slowly for segmented objects and quickly for non-segmented objects. We decided to use this approach since our background does not change often.

Finding the hand has been previously done by finding the largest connected component in an image [1]. Connected component analysis tends to be expensive, and drives methods to narrow the search space. We chose to use a simple median filter to remove non-hand components of the binary image since once again, our background was not too complicated.

2.2. Finger Tracking

Finger tracking can either be posed as a problem of estimation through Kalman Filtering [5] or as a problem of detection after every frame [4]. In [5], Kalman filtering was used in order to reduce the search space for a finger every time. At the resolution we were operating, 320x240, We found that enforcing simple rules on the maximum limits of finger movement was sufficient for the same purpose.

In detection, a shape filtering approach was done by [1], which was in turn inspired by [4]. In these approaches, natural constraints of hand and finger sizes are used to progressively filter out candidate fingertip pixels. We followed suit and further optimized the shape filters to be able to search for fingertip candidates faster (see section 3.3.2).

2.3. Image capture

Pen and paper solutions to image sharing has been an active field of research in HCI. Automated post-it detection systems[8] have been developed using image segmentation. Pen digitizers such as the Anoto system have also been used to capture drawings both digitally and in ink [9]. The Anoto system is attractive since it automatically converts all drawings into digital form. However, it required a special pen as well as special paper to work, thus making it impractical for spontaneous use. Klemmer's work is perhaps the most inspiring as it allows users to use regular paper and pen. We see finger tracking as a way to augment that system by allowing users to write on any surface, such as notebooks and sketch paper.

3. Approach

To build a successful finger tracker, we rationalized that the critical component was speed. In order to successfully track a selected region of the notebook, we enforced a minimum framerate of 10 fps. This number increases as finger speed increases.

3.1. Setup

We created a mount for the webcam (Logitech QuickCam Pro for Notebook) to be placed next to a laptop PC (Windows XP, 1.4 MHz Pentium M) (see Figure 2). Real-time finger tracking was written in C using the OpenCV library. Analysis and algorithm testing was done in Matlab.

Figure 3 shows two types of data sets used for analysis in Matlab: plain engineer's notebook with hand drawings and a mixture of colored pieces of paper. All data was captured at a resolution of 320x240. The rationale is that higher finger tip resolution in the context of pointing and selection is not needed. Real-time testing was performed on a variety of surfaces.

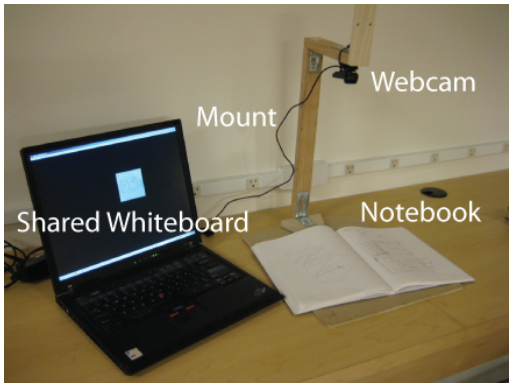

Figure 2. Webcam mount over a notebook, next to a laptop PC.

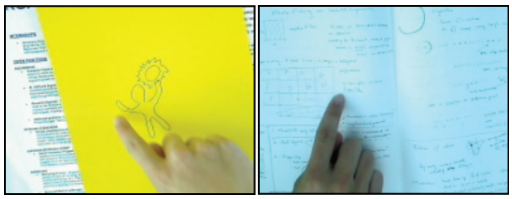
Figure 3. Two types of images used for finger detection

3.2. Hand Detection

3.2.1 Background Subtraction

To detect the hand, we went for a background subtraction strategy. If the background image is a notebook, we can assume that it does not change drastically in shape or color. In fact, most notebooks are white and easily separable from skin. We focused on this approach instead of motion based techniques to satisfy the speed requirements.

Following the CED method, image pixels were converted into a normalized R,B colorspace:

$$R' = R/(R+G+B) \quad (1)$$

$$B' = B/(R+G+B) \quad (2)$$

Hand pixel candidates were detected by thresholding the norm of the difference between the current frame and background frame in this two-channel color space. An alternative method was tried using the YUV colorspace. YUV is designed to separate illumination from color. The 3 channels are constructed as linear combination of RGB values as follows [10]:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad (3)$$
$$U = 0.436 * (B - Y)/(1 - 0.114) \quad (4)$$
$$V = 0.615 * (R - Y)/(1 - 0.299) \quad (5)$$

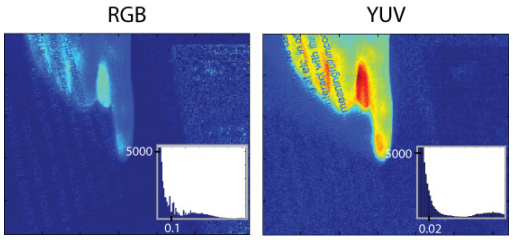

Figure 4. A comparison of RGB and YUV methods of hand segmentation

In Figure 4 we show the results of the CED and YUV methods of background subtraction. We found YUV to be much better in separating hand and non-hand images, as shown by the histogram of differences. It was found however, that in the case of large red objects, the YUV method had severe problems in separating the hands. This phenomenon is also revealed in [2], which operated in the HSV colorspace, which claimed that skin tones have an unusually high red component regardless of pigment.

3.2.2 Histogram based skin classifier

As an alternative hand detector, a skin classifier based on histograms as described in [2] was constructed. Two histograms were formed from image groups that contain skin and that don't. A pixel is binned according to its color value, i.e. each color range corresponds to a certain bin. We restricted the histograms to two dimensions by using the YUV format. This is possible since most color information is contained in two channels [10].

Each pixel was then binned in the skin and non-skin histograms and the ratios were used to classify the pixel as skin or non-skin. The formula used is given in [6], the parameter Θ is used to trade off correct detections and false positives [2]. In our experiment we used $\Theta = 1$.

$$\frac{P(U,V|skin)}{P(U,V|non-skin)} \geq \Theta \quad (6)$$

The results of the classifier are shown in Figure 5. We find that the classifier does an excellent job of finding the hand and leaving out noise. The results, however, are user dependent if the classifier is not trained heavily ahead of time. We chose not to use this to avoid a training sequence with the user's hand at the beginning of a run.

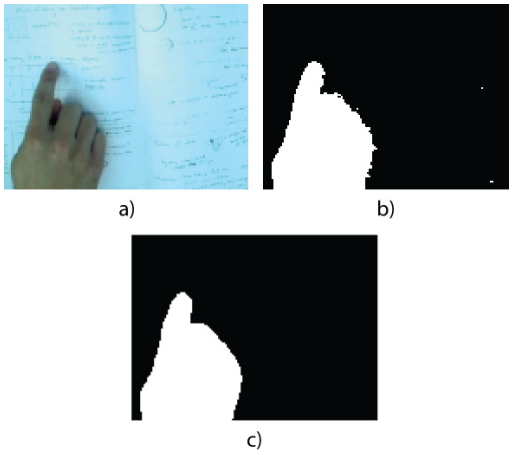

Figure 5. Results of histogram skin classifier. a) original image. b) classified image. c) median filtered

3.2.3 Background Update

It was deemed necessary to have a dynamic background since the user would inevitably write in the notebook. Background updates were performed with the following algorithm [4]:

Background Update Algorithm

1. Determine if a new pixel increased in illumination using the Y-channel.
2. If the illumination increased, calculate a running average with the last 10 background frames.
3. If the illumination decreased, assume it is a new object and calculate the running average with the last 200 frames.

Here the 200 frames correspond to about 10 seconds, an observation done by [4] as the average time a hand will linger in the screen. Step 1 is reasoned by the the fact that hands leaving the picture will reveal pixels of higher illumination, and should thus be updated faster. Pixels that darken tend to be hand pixels and should be updated slowly.

After thresholding, an amount of salt and pepper noise in the image is inevitable. Figure 6 shows the dirty image and the results of median filtering versus finding the largest connected component. Median filtering was done with a

13x13 kernel with the size corresponding to about half a finger width. We found that median filtering did an excellent job of removing extraneous noise as well as filling the holes in the hand. Since a O(1) in kernel size median filtering algorithm is implemented in OpenCV [6], finding the largest connected component was deemed unnecessary and computationally expensive (0.25 seconds in Matlab).

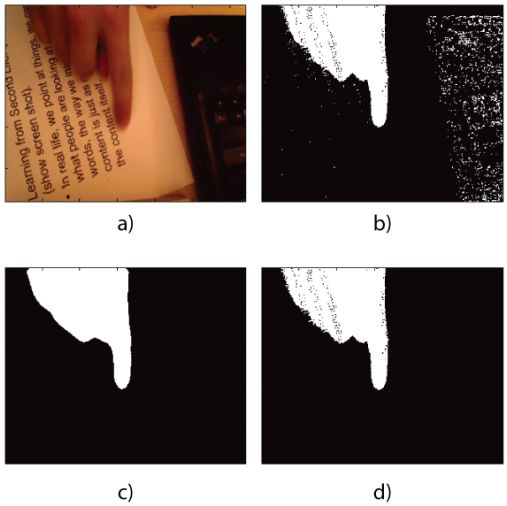

Figure 6. A comparison of median filtering and connected component analysis. a) original image. b) thresholded image c) median filtered d) largest connected component

3.3. Detecting the fingertip

To detect the fingertip from the binary hand image, we tried two different algorithms. Boundary tracking with circular search paths, a home made method, relies on tracking the edge and collecting points along the edge and then feeding these points to a classifier. The second method we tried was a modified form of shape filtering as described in [1].

3.3.1 Boundary tracking with circular search paths

The boundary tracking method operates on images in binary format. This facilitates the logical comparisons needed. Salt and pepper noise can seriously degrade the performance so a median filter is applied prior to it's use.

Start and end points for the algorithm are found by searching the boundary of the image. A starting point is recorded whenever a region of ones, corresponding to white in Figure 8, is entered. To increase robustness against noise, five points with value of one must be encountered consecutively. Similar logic is used to find end points.

Given a boundary point, the next point is found by traversing a circle around that point, until another boundary is found. This is then repeated with the new boundary point. In the upper right of Figure 8 the search path is shown with the starting point on the circle directly above the center point. To keep the circular search path within the image frame, the center point is moved away from the image boundary if it's too close to it. The algorithm collects edge points in this way and stops when the distance to the end point is sufficiently small.

The direction of which the circle is traversed is chosen by observing the value of the starting point on the circle. Using the colors in Figure 8 the direction is clockwise if the starting point is black and counterclockwise if it's white.

This method is very fast since it operates on a small number of image points. The radius of the circle can be varied to increase the number of edge points and hence the resolution in detection of the fingertip.

A classifier is used on the edge points to detect the fingertip. The classifier is described in the following pseudocode and in Figure 7. The classifier used was based on the assumption that the hand entered from the baseline so the finger pointed upwards in the image. It's possible to make it work for all directions of the hand but because of time limitations we restricted the algorithm in this way.

Finger Tip Classifier Algorithm

1. Differentiate the edge points in the y direction
2. take sign of the differentiated signal
3. **for** each 7 consecutive edge point block
   sum the sign signal over the block
   if |sum|==1 and **if** distance between first and last points in block matches finger width → classify point as a fingertip

We found the method to have some drawbacks. It was very sensitive to noise, especially in the vicinity of the hand. Noise pixels caused the method to bounce back and forth between an outlying noise pixel and the hand. The classifier sometimes found more than one candidate for fingertip. In that case the candidate with the lowest y value, i.e. farthest away from the base of the hand, was chosen as a fingertip.

3.3.2 Shape Filtering

The method of shape filtering takes the hand extracted image and runs each pixel through a series of refining conditions for finger detection. We describe the filters as follows:

Shape Filters

1. Pixel is a hand pixel.
2. A circle with radius fingerMin drawn around the pixel is fully connected.

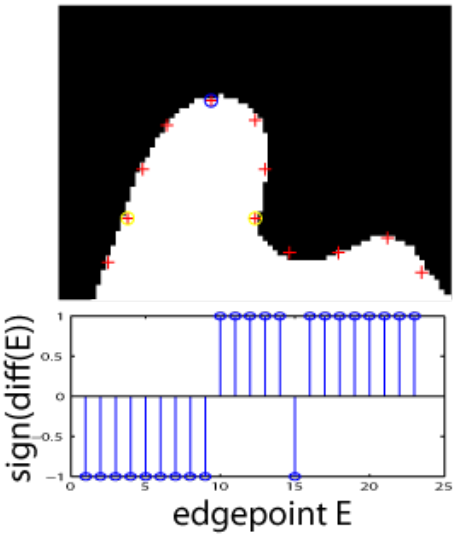

Figure 7. Above: Edge points on finger, fingertip is marked with a circle. The first and last points in the block where the fingertip was found are marked with circles. The distance between these points must fit within a range of possible finger widths. Below: Sign of the derivative of y coordinates of edge points. This signal is used for classification.

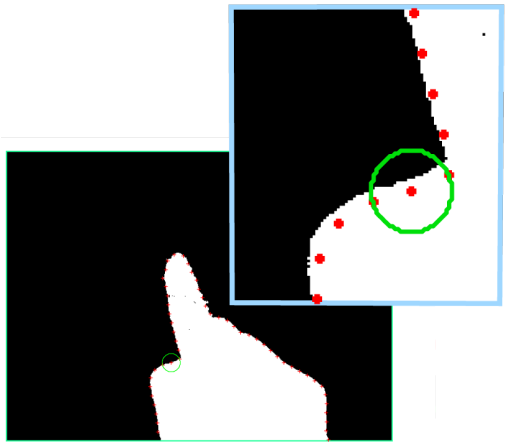

Figure 8. The plus signs are edge points found with the boundary tracking method. Two small circles at the fingertip indicate possible fingertip points. The larger circle on the thumb shows a search path around the point marked by plus at the center of the circle. In the upper right, we show boundary tracking with circular search path.

3. A circle with radius fingerMax drawn around the pixel has only one connected component with length consistent with a normal finger width.

For our application, fingerMin and fingerMax were determined by hand (no pun intended) by placing a variety of hands under the mounted webcam and counting the pixels spanning finger tips. Figure 9 shows the results of successive filters on candidate pixels.

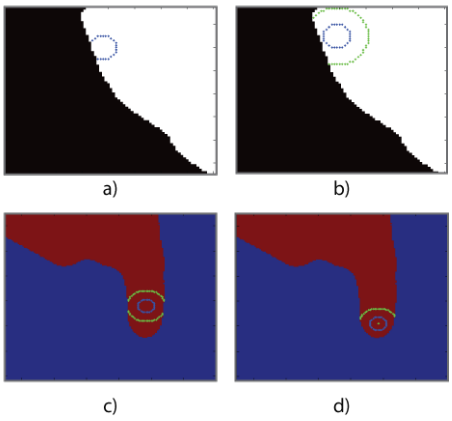

Figure 9. Effects of shape filters on different pixel areas. a) Edge of hand satisfies Filter 2 (blue pixels). b) Filter 3 (green pixels) fails on edge of hand because connected component is too large. c) Filter 3 fails on middle of finger because there are two connected components. d) Filters 1,2,3 satisfied at finger tip.

We reduced the number of pixels searched in [1] by assuming that traversing a small circle around each pixel was enough to establish it as a finger candidate. Letessier [1] searches through all pixels within the circle to ensure it is a fully connected component.

One of the key benefits of using shape filtering is that it is more robust to noise. By progressing through successive filters, shape filtering can also find multiple finger tips, as shown in Figure 10.

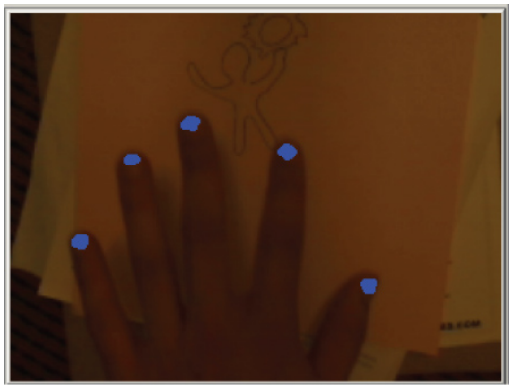

Figure 10. Shape filter running on an open hand. All candidate fingertip pixels are shown in blue.

3.4. Finger point tracking

Inevitably, sometimes a frame would fail to capture a well-segmented hand and the finger tip would be assigned

to spurious noise. To deal with these conditions, we imposed a maximum boundary in which the fingertip could be moved based on hand speed. If the finger tip exceeded these boundaries, the frame was skipped.

4. Results

Our system processed frames at 15 fps. We found this to be an upper limit to the system even when no processing occurred for each frame. This may be due to limitations of our driver support for the webcam.

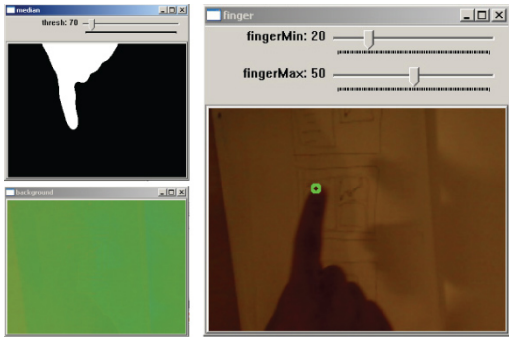

Figure 11. OpenCV application. Updated background in lower left, threshholded hand in upper left, finger detected in right window with sliders for parameters minimum and maximum finger widths.

In our application, we achieved optimal results by first setting the threshold for hand segmentation to match the current environmental conditions. We noted this threshold could fluctuate between .1 to 1 (the norm of the distance between pixels in YUV space).

Below we show a breakdown of times calculated for each of the algorithms developed.

Algorithm times (ms)	
Background Subtraction	10
Median filtering	10
Edge point classifier	1
Shape Filtering	30

In Figure 2 we showed the webcam mount setup. We found that the wooden platform of the mount had colors very similar to skin tones and made hand separation very difficult. Once this setup was covered with the notebook, a proper threshhold could be chosen to reliably segment the hand using the YUV color scheme.

5. Discussion

We found that the primary hurdle in developing a successful finger tracker was a clean segmentation of the hand. Once a clean binary image of the hand is produced, finger detection is trivial using either of the methods addressed above. For future work, we envision a more adaptive approach to determining the hand, potentially by continuously updating a color histogram of the hand after detection. Our investigation into histogram based skin classification shows that this method would be very robust to noise. Since our system would be used by one user for an extensive period of time (a teleconference session is between 30 min to 2hrs long), the system would be able to stabilize by learning the user's hand.

The overall frame rates we were achieving, even with the more extensive shape filtering algorithm, indicate that we can perform more computation to isolate the hand successfully.

References

[1] J.Letessier, F. Bérard . 2004. *Visual Tracking of Bare Fingers for Interactive Surfaces.* UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology. 1, 2, 4, 5

[2] M. Jones, J. Rehg. 1999. *Statistical color models with application to skin detection.* IEEE Conf. on Computer Vision and Pattern Recognition. 1, 3

[3] R. O'Hagan, A. Zelinsky. 1997. *Finger Track - A Robust and Real-Time Gesture Interface.* Advanced Topics in Artificial Intelligence, Tenth Australian Joint Conference on Artificial Intelligence (AI'97) Proceedings.

[4] Christian. Hardenberg and F. Brard. 2001. *Bare-hand human-computer interaction.* In Proceedings of Perceptual User Interfaces. 2, 3

[5] A. H. Sayed, T. Keaton, S. M. Dominguez. 2001. *Robust Finger Tracking for Wearable Computer Interfacing.* ACM International Conference Proceeding Series; Vol. 15. 2

[6] S. Perreault, P. Hebert Sept. 2007 *Median Filtering in Constant Time* Image Processing, IEEE Transactions on , vol.16, no.9, pp.2389-2394. 4

[7] W. Tsang, P. Kong-Pang, December 2005. *A finger-tracking virtual mouse realized in an embedded system.* Intelligent Signal Processing and Communication Systems, 2005. 1

[8] S. Klemmer, M. W. Newman, R. Sapien, 2000. *The designer's outpost: a task-centered tangible interface for web site information design.* CHI '00 Extended Abstracts on Human Factors in Computing Systems. 2

[9] R. Yeh, S. Klemmer, A. Paepcke. 2007. *Design and Evaluation of an Event Architecture for Paper UIs: Developers Create by Copying and Combining.* Stanford University Computer Science Department Technical Report. 2

[10] YUV, (2008, March 13). In Wikipedia, The Free Encyclopedia. Retrieved March 21, 2008, from http//en.wikipedia.org/w/index.php?title=YUV&oldid=197978234 3