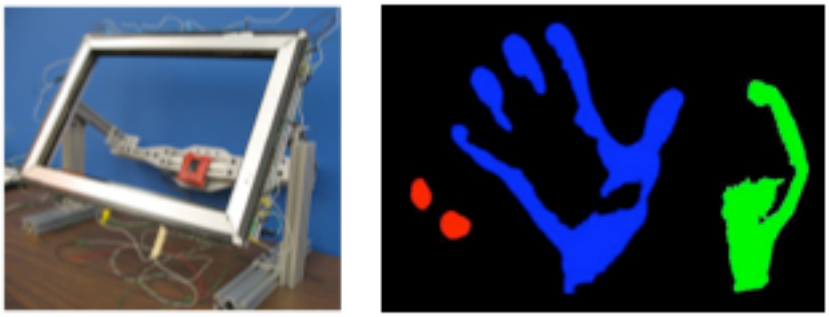
Classifying Hand Shapes on a Multitouch Device

Daniel Gibson, Aaron Rosekind, Ashley Wellman
Stanford University, Stanford, CA 94305
Email: {gibsnson, rosekind, wellman}@stanford.edu



Our custom multitouch device, examples of classified hand shapes

1 Introduction

As modern computer systems become more complex, the latest tablet laptops, PDAs, and handheld gaming devices are all moving towards more intuitive and dynamic touch interfaces. Multitouch has sparked significant interest amongst academic researchers and hobbyists alike. However, standard open source libraries, the most popular being Touchlib by NUI Group, do not distinguish between different hand shapes. The ability to distinguish between different types of contact would allow such intuitive gestures as collecting objects by cupping with the sides of the hand, or erasing by wiping the screen with the palm of the hand. Our goal, then, is to create a system that will correctly distinguish between different hand shapes. This is a difficult problem, because the size and orientation of a given hand shape can vary greatly.

Our project solves this problem by extracting relevant features from a preprocessed hand image and applying machine learning algorithms to identify different kinds of hand contacts. Our system can identify the fingertip, the side of the hand, and the palm of the hand, despite widely varying hand sizes and orientations.

2. Hardware

The first stage of our project was to build a light-based multitouch surface. Light-based multitouch devices involve three main components: a camera with an IR filter, an IR light source, and a touch surface. Contact with the touch surface scatters IR light towards the camera, causing the camera to register a bright spot at the point of contact. This technique can be implemented using a variety of methods, which are distinguished primarily by their method of illumination. We tested three such methods: rear diffused illumination (rear-DI), frustrated total internal reflection (FTIR), and LED light-plane (LED-LP). Rear-DI uses IR LED panels to bathe the touch surface with light from below such that the light is reflected back by objects above the surface. FTIR uses IR LEDs around the perimeter of the touch surface. The light from these side-mounted LEDs is directed into the surface from the sides, and reflects internally until a

contact scatters light towards the camera. LED-LP uses the same hardware setup as FTIR, but a thinner surface is used, creating a light-plane above the surface which scatters upon contact.

The question of which method to choose depends heavily on the intended use of the device. We identified three primary criteria for comparing the different methods: overall brightness of IR image, image contrast between touched and untouched areas, and whether a part of the hand hovering just above the surface is incorrectly registered as touch. Rear-DI has the high brightness, low contrast, and worst presence of hover. FTIR has very low brightness, high contrast, and no hover. LED-LP lies between these two extremes. Sample raw camera images from the three methods are shown in Figure 1.

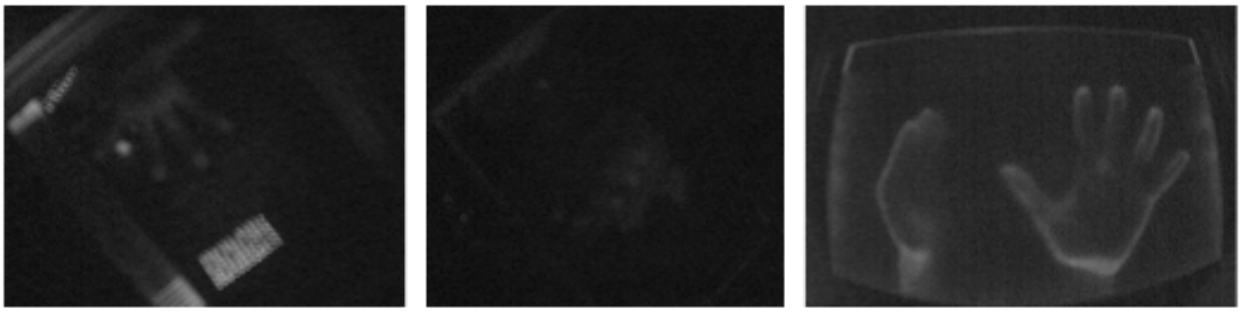


Figure 1: Raw IR images captured using (from left to right) rear-DI, FTIR, and LED-LP

After testing all three, we found the method that worked best for this project to be LED-LP, since the hover range for rear-DI was too high and the brightness of FTIR was too low. The LED-LP we built only had a narrow band in which hover was an issue, and had much higher brightness than our FTIR.

3. Image Processing

Substantial pre-processing is neccesary in order to transform the raw data from the multi-touch device into useful input for our classifier. First, we use background subtraction to remove the light gradient caused by light leakage around the touch surface. Next, we apply a Gaussian blur to remove noise, followed by a threshold filter that transforms the gray-scale image into a binary image. Noise is further reduced by removing all blobs with area less than 80 pixels. Finally, we use a standard blob detection algorithm to divide the filtered image into zero or more distinct blobs.

In order for our classifier to act on hand shapes rather than individual blobs, we aggregate related blobs into groups. The importance of this aggregation is illustrated by the palm, which is composed of several distinct blobs but must be classified as a single image. Our blob aggregation algorithm begins by considering each of the blobs to be separate, and groups any two blobs whose convex hulls intersect. The algorithm then proceeds recursively, with two groups being combined whenever their convex hulls intersect, and runs until a steady state is reached. This algorithm is exhibited in Figure 2.

Figure 2: Blob aggregation algorithm

Our final pre-processing step standardizes the orientation of the grouped blobs. We perform linear regression on the image to find the line of best fit for the image, and then rotate the image so that this line is vertical. This final image is used as input for our classifier.

4. Classification

We used the Spider machine-learning Matlab toolbox to train an SVM that attempts to classify the processed images into one of three classes: fingertip, side of the hand, and palm. Several distinctive features of the image, outlined in Table 1, were used as input to this classifier. We also experimented with performing Naive Bayes on the pixel data, but abandoned it after our SVM gave us such good results.

We created a training data collection program that captures camera images, extracts the desired features, and saves them with a specified label. Using this program, we created a training set comprised of 235 raw images, with one example contact in each image. We collected 50 finger examples, 119 side examples, and 66 palm examples, with data collected from 3 different users. Additional side examples with a variety of orientations were used, because initial results suggested that the classifier had the most difficulty identifying sides.

Features Used
Major Axis Length
Centroid (1,2)
Area
Filled Area
Perimeter
Solidity
Convex Area
Equiv. Diameter
Minor Axis Length
Centroid (1,1)
Extent
Eccentricity

Table 1

We optimized our SVM by performing model selection on kernel parameters and selecting the kernel with the lowest cross-validation error. Model selection significantly decreased our error. For the polynomial kernel, the degree parameter was tested between 2^-5 and 2^20, with an error range from 20% to 73%. For the Gaussian kernel, the sigma parameter was tested between 2^-5 and 2^20, with an error range from 0% to 70%. We chose the Gaussian kernel with a sigma value of 2^10, since the optimized Gaussian kernel achieves a 10-fold cross validation error of 0% compared to the optimized polynomial kernel error of 20%. A plot of the cross-validation error for the Gaussian and polynomial kernels as the model parameters vary is shown in Figure 3.

We found that performing backwards search feature selection allowed us to eliminate some extraneous features while still maintaining 0% cross-validation error. Using a Gaussian kernel with sigma = 2^10, we found five extraneous features: equiv. diameter, minor axis length, the x-coordinate of the centroid, extent, and eccentricity. Figure 4 shows the error as each feature

identified by backwards search is added back to the classifier. The most useful feature is major axis length, which reduces error to 27% even when it is the only feature.

Figure 3: Model Selection

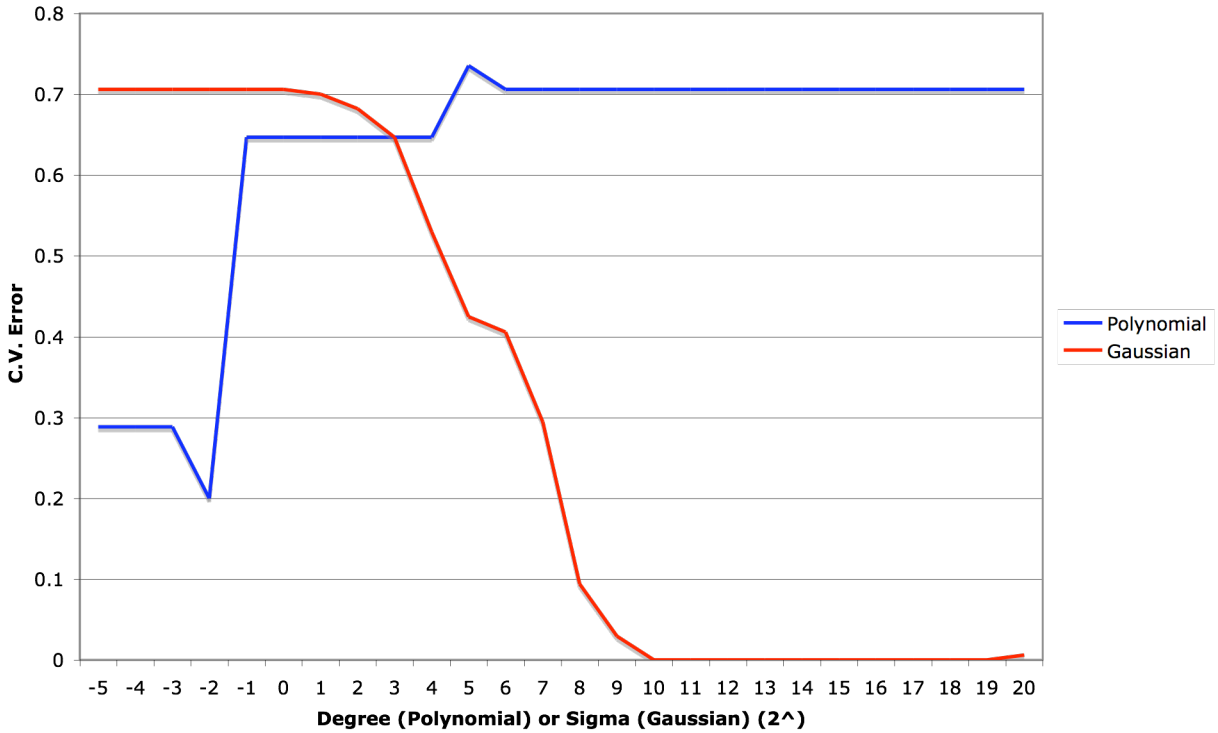
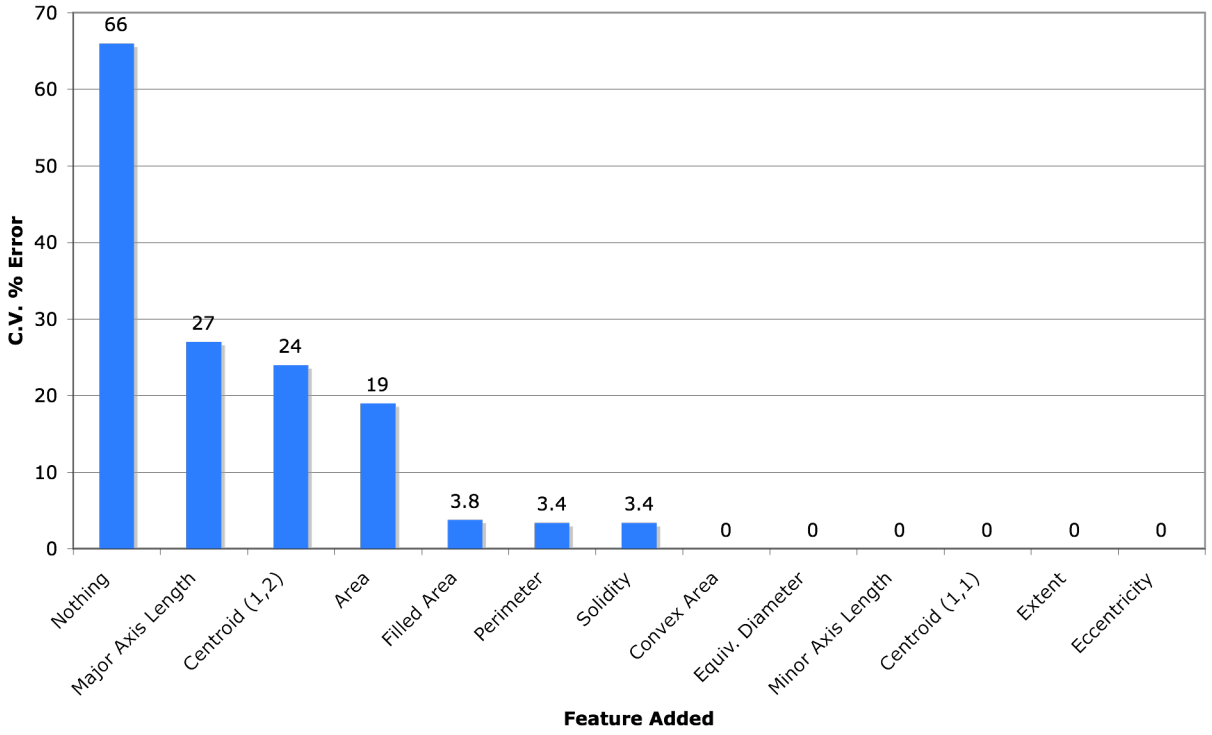


Figure 4: Feature Statistics



5. Conclusion and Future Work

Our SVM allowed us to successfully classify hand shapes on our custom-built multitouch device, as our classifier achieved 0% 10-fold cross-validation error on our training data set of 235 images. We also created a demo program that allows users to touch the screen and have their hand shape classified. Tests of this tool during a live demo of the project were very successful. While we did not collect hard data during our live demo, most hand gestures were classified correctly. The main source of error during our demo was with users wearing long sleeves or having hands bigger than any of our training set. Additional training data with sleeves and a wider variety of hand sizes would likely help to prevent this problem.

Despite the fact that we built several iterations on our hardware, there remains much room for future improvement in this area. For example, a refined FTIR or rear-DI set-up with high touch/hover contrast would yield filled shapes of contacts, rather than just the outlines of contacts from LED-LP. With these filled shapes, our blob grouping algorithm would no longer be necessary and classification would require less image processing time.

Additionally, while our system can perform classifications in a few seconds, it would become much more useful for most applications if classification could be done in real-time. Moving the system from Matlab to C++/OpenCV would increase our computational speed and facilitate integration with future projects involving multitouch classification.

Once classification can be done in real-time, the problems of contact tracking and gesture classification should be addressed. Contact tracking requires maintaining a list of contact locations, shapes, and heading, and can likely be accomplished with only minor extensions to currently existing blob tracking algorithms, as are currently used by Touchlib. Such algorithms compare successive images, determining whether a contact is a new contact or a translation of a current contact and removing old contacts from the list. Gesture classification would involve not only detecting hand shapes, but also identifying different hand motions upon the touchscreen.

6. References

J. Weston, A. Elisseeff, G. Bakir, and F. Sinz. "Spider: General Purpose Machine Learning Toolbox in Matlab." The Spider. 24 July 2006. 12 Dec. 2008 <http://www.kyb.mpg.de/bs/people/spider/>.

N. Motamedi, et al. "LCD FTIR - first results." Touchlib Forum. 23 Nov. 2007. NUI Group. 12 Dec. 2008 <http://nuigroup.com/forums/viewthread/1040/>.