Gesture Recognition with Applications

[CSE 634 Class Project Report]*

Oleksiy Busaryev
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210, USA.
busaryev@cse.ohio-state.edu

John Doolittle
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210, USA.
doolittle.15@osu.edu

ABSTRACT

The focus of this project is recognizing hand gestures captured from a webcam in real-time, and then using the gestures to control real-world applications (in our case, Google Earth and the TORCS racing game simulator). Background subtraction and HSV-based extraction are compared as methods for getting a clean hand image to analyze. The gesture in each hand image is then determined with Hu moments or a local feature classifier, and each gesture is mapped to a certain keystroke or mouse function. Development is performed in C++ using OpenCV computer vision library to allow for real-time performance.

1. SETUP

A standard webcam with 640x480 resolution was used. The resolution proved more than needed for analyzing gestures of one hand, so frames were captured at half resolution (320x240) to improve processing speed and to make sure that our approach is able to handle low-resolution images. The webcam was placed between one and two feet above a desk, pointing down toward the desk's surface. Various lighting conditions were used during development; the fluorescent ceiling lights in Dreese Laboratories and one, two, or three lights during all times of day and night in a windowed room were tested.

Problems encountered. The windowed room had more noise issues than the room in Dreese. This was not surprising because of the widely varying lighting. It was generally acceptable but very difficult to get a clean image out when the lighting was exceptionally bad. In addition, incandescent light bulbs used in the room produced yellowish light, which made hue-based extraction more difficult. See Figure 1 for a frame captured in Dreese under fluorescent lighting.

*Additional information, including videos, can be obtained at the project website `http://cse634.wikidot.com`

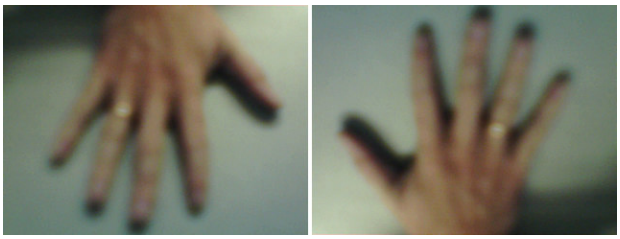

Figure 1: Captured frame. The camera is placed in front and above the hand, so the image has to be flipped vertically and horizontally.

2. HAND IMAGE EXTRACTION

2.1 Background Subtraction

Background subtraction is the first approach to extracting a hand that was attempted. It is a very simple approach that is highly susceptible to errors from noise, so the first fifty frames of the video feed were accumulated as an entire background sequence to improve robustness over using a single frame.

Figure 2: Mean background image. The hand was only present for a few frames and thus nearly averaged out by the rest of the static background.

Figure 2 demonstrates the advantage of using a background sequence over a single frame. Often a scene will have objects moving through it that are not supposed to be part of the background. By taking the mean of many frames, these objects can be averaged out and only the static background will remain (notice how transparent the hand appears compared to the paper edge). The mean and variance of each pixel through this initial sequence is calculated and used to find the Mahalanobis distance between the sequence and any new frame. If the distance exceeds a threshold then

the new pixel is sufficiently different from the background and is turned on in the final binary hand image. A slider was used to test different thresholds during development and ultimately a threshold of 20 was used for the final code. Finally, a median blur is performed to remove salt and pepper noise. Dilation and erosion were experimented with to further clean up the image but were not kept in the final code; this approach had such significant problems with shadow that it seemed pointless.

Problems encountered. Shadows made this approach essentially useless for the purpose of gesture recognition, at least when compared to HSV. While good images were occasionally possible, generally the shadows overwhelmed the image. This essentially caused two overlapping hands to be extracted, which resulted in a blob that could not easily be analyzed (see Figure 3)

Figure 3: Hand contour with errors from shadow.

2.2 HSV-based extraction

The second hand image extraction technique we tried is based on the fact that human flesh extraction is much easier in the HSV (hue-saturation-value) color space than in the RGB color space [4]. It turns out that flesh has distinct hue value which allows to separate the hand from the rest of the image by simple thresholding.

Webcams are known for low image quality (especially in low light conditions), so in order to combat color noise, we ran Gausssian smoothing before converting to HSV color space (see Figure 4 for an example). Median filtering seems to be a better choice, but not in the case of heavy noise.

Figure 4: Left: after applying 3x3 Gaussian filter. Right: after converting to HSV color space.

So far, we cannot see the benefits of HSV. But if we look at the hue component separately (see Figure 5), we can see that hand noticeably stands out as a light-gray object on the dark background. So, we simply threshold hue component. Human flesh in our case roughly corresponds to the $[90, 140]$ interval, however, this seems to depend on lighting type (incandescent or fluorescent).

Figure 5: Left: hue component of the image. Right: thresholded hue component ($90 \leq$ hue ≤ 140).

We see that significant binary noise is still present in the image. To combat the noise, morphological operations are employed. Regular open and close operations by themselves do not give us significant improvement when the noise is large, so we perform 3-fold 'open' operation (3 erosions followed by 3 dilations), then 3-fold 'close' operation (3 dilations followed by 3 erosions) instead. This approach efficiently gets rid of the noise (see Figure 6. Finally, we use the hand image as a mask for initial captured frame (we perform logical AND'ing of both images).

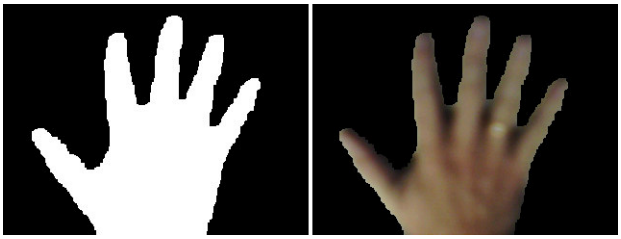

Figure 6: Left: image denoised using morphological operations 'open' and 'close'. Right: original image masked by hand image.

In general, HSV-based extraction seems to give very good results with minimal artifacts.

Problems Encountered. Hue-based approach has certain drawbacks: if extra objects having the same hue as a flesh are present in the scene, they are sometimes recognized as flesh (see orange scissors in the Figure 7 below). Saturation should be carefully adjusted in this case.

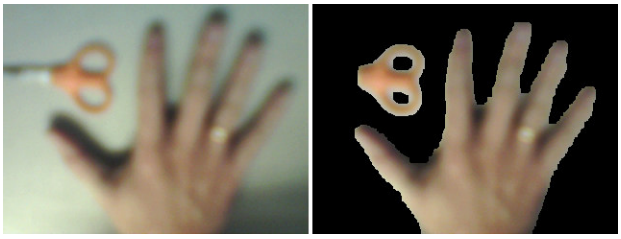

Figure 7: Left: captured image with extra object in the scene having the same hue as a hand. Right: final image with object present.

In addition, when incandescent lighting is used (especially of low intensity), it may give yellowish tint to some objects, which causes erroneous recognition them as flesh.

3. GESTURE RECOGNITION

3.1 Local Feature Classifier

The local feature classifier is based on the work by Kumar et al. [3]. The contours of the hand image are found. Then the largest contour is selected as the hand outline and curvature measurements are taken between points on the outline. The measurement used is the angle formed between the vectors of three points (the middle point is the head of one vector and the tail of the other). The points on the outline are too close together for adjacent points to give meaningful results, so the calculation is done between points that are spaced further apart. Spacing of 1 to 255 points along the contour between the middle point and the other two were tested, and 19 seemed to give good results. It is possible that further refinement to use less spacing could result in improved accuracy, but the potential advantage of this was not seen during testing. The extrema of the angles are then found. Setting a threshold to capture the tip of one finger without eliminating the tip of another proved surprisingly difficult; just taking the top nine angles does not necessarily correspond to five finger tips and four gaps between fingers. Because of this, multiple extrema on the same finger are accepted. The sign of the angle is then used to classify the point as either a peak or a valley. A positive sign corresponds to a peak, and a negative sign to a valley. Different combinations of the count of peaks and valleys correspond to different gestures. The numbers of peaks and valleys are stored and averaged across frames in an attempt to smooth recognition errors. The Figure 8 shows the five gestures that can be recognized fairly consistently. The words on each image refers to how the gesture was mapped to the mouse (see Section 5.2).

Problems encountered. The largest problem during development was finding a good way to threshold the peaks and valleys. The classifier could potentially be more powerful if it were known that one finger pointing always returned one peak, but instead a range of peaks was settled on. While good for two or three gestures, the classifier struggled beyond that because of this wider range. 'Right Mouse Button' and 'Wheel Down' were often misclassified as each other, for instance. This is not very surprising because they do look similar in terms of peaks and valleys, but recognizing five gestures does not seem unreasonable if improvements could be made. Another issue is that the classifier is fairly sensitive, even after attempts at smoothing the count of peaks and valleys across frames. Random noise could cause the hand contour to shift in such a way that a meaningless area would be recognized as an extreme angle. Also, a slight shift in the hand could cause a fingertip to be more or less of a sharp angle, which could cause peaks to be marked or lost. Overall the recognition tended to jump between gestures too rapidly. More consistent, steady recognition would have gone a long way towards usability.

The gesture is recognized as 'Clear' when the number of peaks plus the number of valleys is less than two (it is usually zero but some room for error is used). 'Wheel Up' is when there are more than 35 peaks and more than 30 valleys. 'Wheel Down' occurs when the absolute value of the difference between the number of peaks and valleys is less than two (i.e. the number is almost the same because one area of peaks and one area of valleys gets detected). Note that for this case the hand must be angled carefully so as to make two of the four fingers appear broad enough along to the top to not get recognized as having peaks. 'Left Mouse Button' is when the number of peaks is greater than seven and there are zero valleys (the thumb must be at nearly a ninety-degree angle to the hand for it to not have valleys). If none of these conditions are met, then 'Right Mouse Button' is checked by testing if the ratio of peaks to valleys is between one and three. Otherwise the classifier returns -1, indicating that no gesture could be matched. The numbers used to classify each gesture were determined by watching for pattern in the count of peaks and valleys while performing various hand movements.

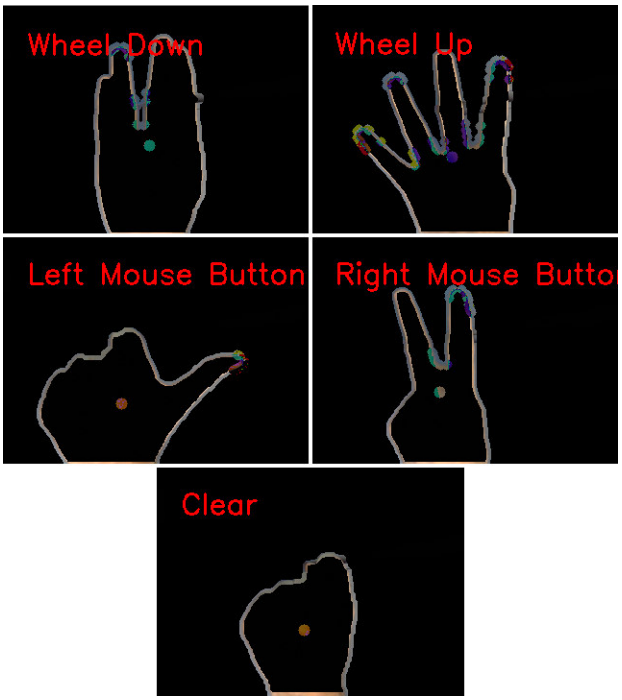

Figure 8: The five gestures recognized by the local feature classifier.

3.2 Hu Moments

In this approach, we follow the technique used by Bobick et al. [1] for comparison of motion history templates. First, we compute statistical models of several hand gestures. For each gesture, we collect significant number of training images (around 100 per gesture). We compute seven rotation- and scale-invariant Hu moments [2] for each training image. This seven-number code serves as a brief description of an image. To build the statistical model of each gesture, we compute the mean and the covariance matrix across the corresponding set of training images' descriptions.

For gesture recognition, we compute Hu moments of the captured hand image. Then, we calculate Mahalanobis distance from computed moments to each gesture. The gesture minimizing Mahalanobis distance gives us a match.

Collecting around 100 training images per gesture, we were able to get a database that is enough for reliable and efficient recognition of several typical hand gestures (see Figure 9).

See example: `www.youtube.com/watch?v=a6_qJTX18DI`.

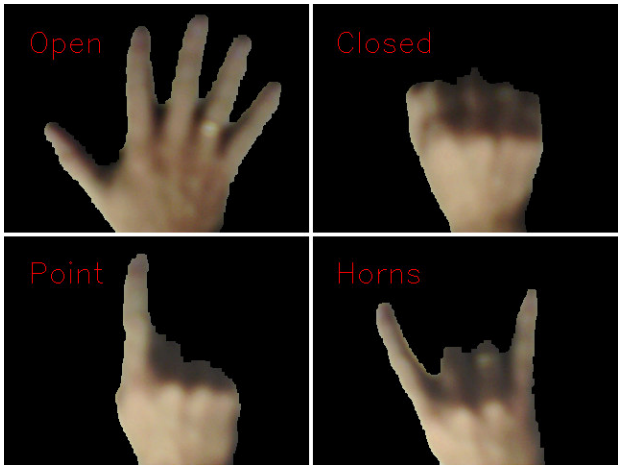

Figure 9: Four gestures succesfully recognized by statistical matching with Hu moments.

However, this approach definitely suffers from obvious drawbacks. Seven Hu moments by no means give a complete description of an image, hence are suitable only for giving very rough estimation of possible match. For instance, quite often we were not able to distinguish between 'Closed' and 'Flat' gestures (see Figure 10). Probably, scale invariance of Hu moments is the reason.

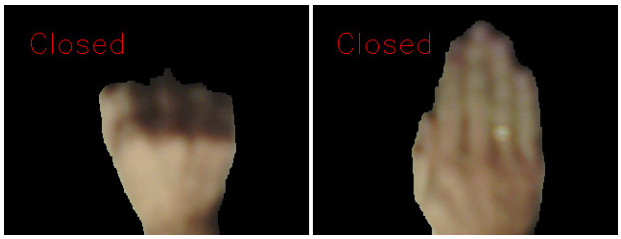

Figure 10: Not always we can distinguish 'Closed' gesture (left) from 'Flat' gesture (right).

In addition, it is almost impossible to detect variations in finger positions and mutual distances: we can tell apart 'Open' and 'Close' gestures, but not 'Open' and 'Four-fingers-open'.

Problems encountered. Several unexpected technical problems arose while implementing this approach. Moment descriptions of gestures vary in order of magnitude, so some kind of normalization was needed. We simply divide the Mahalanobis distance by the maximal distance of training images to the mean for a particular gesture. To get reasonable matching accuracy, number of training image should be large: less than 100 per gesture is not enough. What is more important, we should capture not only ideally-looking, emphasized gestures, but vague, indistinct as well; otherwize, for instance, half-straightened pointing finger will not be recognized properly. Once sufficient number of 'almost-right' images is collected for each gesture, matching works nice.

In addition, there were some technical issues that took time to fix. Covariance matrices computed by OpenCV are not normalized by default, so *CV_COVAR_SCALE* parameter should be passed to the *calcCovarMatrix* function. Instances of image matrix class are reference counted, so *clone* function should be called whenever deep copy is required.

4. ORIENTATION RECOGNITION

Hand orientation recognition is needed to provide direction control for the racing game. Our first idea was to imitate steering wheel rotation with both hands. However, in this case the camera should be pointed at the user, which brings in heavy dependency on lighting conditions and clothes that user can wear. So, instead, we switched to controlling the car movements with the pointing finger; finger direction is mapped to the 'left' and 'right' control keys. Brakes and acceleration are controlled by moving the hand backward or forward, respectively.

So, there are two issues to be addressed:

- determine the exact position of the pointing finger tip;
- fit a line into a pointing finger.

4.1 Finding the Tip of the Pointing Finger

We extract the contour of the hand using the findContours OpenCV function. Most reliable results are achieved with the *CV_CHAIN_APPROX_TC89_KCOS* parameter, which enables Teh-Chin chain approximation algorithm [5]. See Figure 11 for an example.

Figure 11: Left: hand image. Right: contour extracted from the hand image.

Next, we have to find the exact position of the pointing finger tip. The first approach we tried was based on curvature estimation. After approximating the contour with a sparse enough polygon, we used central differences to compute first and second order derivatives. Then, curvature at each point can be estimated by the following formula:

$$\kappa = \frac{|x'y'' - x''y'|}{(x'^2 + y'^2)^{\frac{3}{2}}}$$

. The maximal curvature point was chosed as the finger tip. However, results we obtained were not that impressive, since contour was not smooth, so curvature estimation was affected by noise. Higher (fourth) order differences did not give observable improvement.

See example: `www.youtube.com/watch?v=NqEiDP60j0w`.

So, instead, we chose simpler approach: for each contour point p_i, we simply compute the angle between vectors $p_i p_{i+d}$

and $p_i p_{i-d}$, where d is chosen experimentally (8-10 in our case). Then, finger tip is selected as the point minimizing this angle. This gave us sufficiently reliable finger tip position estimate (see Figure 14).

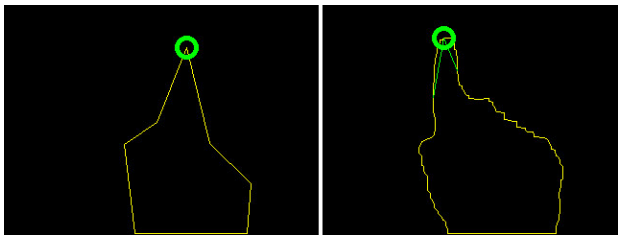

Figure 12: Left: finger tip extracted using curvatures. Right: finger tip extracted using angles.

See example: `www.youtube.com/watch?v=MUpNaKDqop8`.

4.2 Fitting a line into the pointing finger

The most intuitive way to fit a line into a finger after computing its tip as a minimizer of angle $\angle p_{i-d} p_i p_{i+d}$ would be just to find a bisector of this angle (see Figure 13.

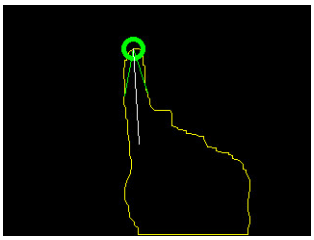

Figure 13: Fitting a bisector.

However, sampling of the contour affects accuracy; this is not important for the finger tip, but crucial for the bisector. As a result, we observe jerky movements.

See example: `www.youtube.com/watch?v=CjGkRexP-BM`.

The next thing we tried is the least squares fit for the points $p_{i-d}, \ldots, p_{i+d}$ (see Figure 14). Surprisingly, results look even worse!

See example: `www.youtube.com/watch?v=w2KnhQXLUUo`.

The key problem seems to be the choice of point sample that is used for line fitting. $p_{i-d}, \ldots, p_{i+d}$ that we use both for bisector and least squares, is not the best option.

So, instead, we chose all the points that are within certain distance of the finger tip as a sample (see Figure 14). In our case, distance $R = 80$ seems to be optimal. Both bisector and least squares give decent results with this approach.

See example: `www.youtube.com/watch?v=Xz4aY68H_BI`.

Problems encountered. The key problem is an adequate choice of sample points representing the finger. Our approach is simple and sufficiently reliable, but something completely different could be tried like computing medial axis.

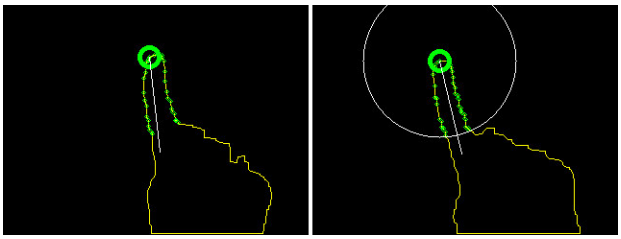

Figure 14: Left: LSQ fit with $p_{i-d}, \ldots, p_{i+d}$ points. Right: LSQ fit with points within $R = 80$.

5. MAPPING TO CONTROLS

5.1 Keyboard

We just used standard Windows ::SendInput function for emulating key press and key release events. Whenever a gesture is recognized by OpenCV, an appropriate key press event is triggered if the corresponding key is not already pressed; if it is, ::Sleep function is called in order to return control to the application for a certain time. Whenever a key press event is sent, all other pressed keys receive the key release event, so no two keys can be pressed simultaneously by gestures ('Point' gesture is an exception, since it is mapped to four different keys: left, right, up and down, in order to provide control over the racing game).

Problems encountered. The only technical problem we encountered was the issue of message passing between the OpenCV application and the racing game application. Simulation of key click events was simple, but key holding required special handling by recurring suspending of our application thread using timeouts.

5.2 Mouse

The Windows SendInput function is used to send left/right clicks and wheel scrolling. When gestures for left or right click are made, a click down event is fired. The click up event is not sent until "Clear" or an unknown gesture is detected. This simulates holding the button down, which is necessary for actions such as dragging something. Cursor movement is done with the help of the hand's centroid. The point in the image (imageWidth * 0.5, imageHeight * 0.75) is used as a 'home' point (the exact center was not used because this approach better centers the fingers). The difference between the centroid and the home point is found each frame, and the cursor is moved in the direction of the vector formed between the home and centroid. To prevent an overly jumpy cursor, a dead-zone is used around the home point. If the centroid is still inside the dead-zone then movement of the cursor is ignored. After testing various sizes, the dead-zone was ultimately set to a 40 pixel circle around the home point. This was large enough for a gesture to not significantly move the centroid while still allowing for sufficient room in the frame where movement is possible.

Problems encountered. The cursor movement is dramatically better than initial attempts, but is still not very natural. Precise clicking is difficult because slight noise can shift the cursor if outside the dead-zone. Also, while the dead-zone does help to mitigate unintentional centroid shifts, it only works if the user returns to the home position to make a new gesture, which can be awkward.

6. OUTCOMES

HSV extraction was used over background subtraction for all of the final gesture recognition because the results were so much better for this application. Keyboard and mouse mapping worked fairly well. Both the local feature classifier and the Hu moments were able to recognize gestures with decent accuracy in real-time. It was enough to control the programs that were part of the initial idea. The problems pointed out in Section 5.2 were not completely eliminated, but Google Earth was successfully controlled with the local feature classifier and mouse simulation (see Figure 15). The feature classifier was translation, rotation, and scale invariant. Testing it with either left or right hand, or even with a different person, did not seem to affect its accuracy.

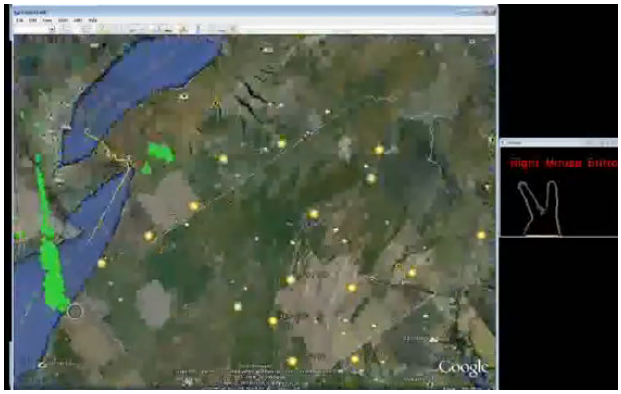

Figure 15: Controlling Google Earth with gestures.

See example: `www.youtube.com/watch?v=y54c1tN0uck`.

Finger line fitting was also successfully used to control TORCS by simulating key press and release events (see Figure 16). We were able to achieve smooth movement of the car (unlike the first version, where we were simulating only key click event). Moving the hand forward and backward was used to simulate acceleration and brake, and moving the pointing finger was mapped to 'left' and 'right' actions.

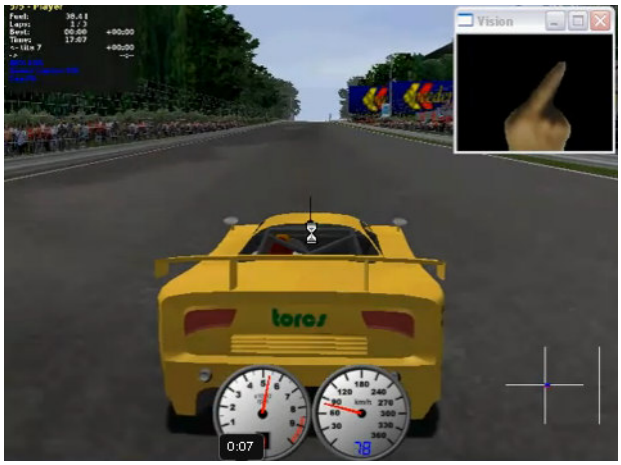

Figure 16: Controlling TORCS with gestures.

See example: `www.youtube.com/watch?v=pgwZBdK23W0`.

7. LESSONS LEARNED

Although background subtraction has potential, it simply did not work well because lighting the scene so as to eliminate shadow is not practical. HSV could have issues with other objects in the scene that have a similar hue and saturation, but it was still much better in a practice. Hu moments provided more consistent results across frames than the sensitive feature classifier. The moments were more of a rough estimation, however, and were not as good as the classifier at distinguishing small differences, such as slightly open fingers. The curvature of the hand did not seem to be reliable enough. It was too smooth, even around points of high curvature, to get a single good point out most of the time. Other methods are better when precise tracking is needed, such as the bisector.

8. ALLOCATION OF WORK

Oleksiy Busaryev worked on HSV extraction, Hu-based matching, fingertip extraction and line fitting, and keyboard simulation. John Doolittle worked on background subtraction, the local feature classifier, and mouse simulation.

9. FUTURE WORK

Noise remained an issue throughout testing, so finding a better way to extract the hand would be very useful. Specifically, a histogram-based HSV extraction could be tried. The classifier lacked precision, so improving peak and valley detection is also a goal. One way of doing this could be through exploring OpenCV methods for getting the hull around the hand and then detecting convexity defects that correspond to fingers. A larger Hu moment database could also be interesting because current testing has been done with very limited samples. Also, work was mainly done in a fairly constrained setting. Trying to deal with less ideal settings, such as one that could result in partially occluded gestures, could be very interesting.

10. REFERENCES

[1] A. F. Bobick and J. W. Davis. The recognition of human movement using temporal templates. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23:257–267, March 2001.

[2] M.-K. Hu. Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2):179–187, February 1962.

[3] S. Kumar and J. Segen. Gesture based 3d man-machine interaction using a single camera. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems - Volume 2*, ICMCS '99, pages 9630–, Washington, DC, USA, 1999. IEEE Computer Society.

[4] N. Tanibata, N. Shimada, and Y. Shirai. Extraction of hand features for recognition of sign language words. In *In International Conference on Vision Interface*, pages 391–398, 2002.

[5] C. H. Teh and R. T. Chin. On the detection of dominant points on digital curves. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11:859–872, August 1989.