Recognizing Hand Gestures with Microsoft's Kinect

Matthew Tang
Department of Electrical Engineering
Stanford University

***Abstract*—In this paper, we propose and implement a novel method for recognizing hand gestures using rgb and depth data from Microsoft's Kinect sensor. Our approach involves looking at specific hand motions, in addition to full body motions, to assist in the recognition of more refined gestures. With this approach, we are able to recognize 'grasp' and 'drop' gestures with over 90% accuracy.**

***Index Terms*—gesture recognition, Microsoft Kinect, vision systems**

I. Introduction[1]

The advent of relatively cheap image and depth sensors has spurred research in the field of object tracking and gesture recognition. One of the more popular devices used to do this type of research is Microsoft's Kinect, which has sensors that capture both rgb and depth data. Using similar data, researchers have developed algorithms that not only identify humans in a scene, but perform full body tracking; they can infer a person's skeletal structure in realtime, allowing for the recognition and classification of a set of full body actions (e.g. [2], [3]).

In this paper, we use data collected from a Kinect sensor to explore the feasibility of gesture recognition on a smaller scale. Instead of attempting to recognize full body actions (e.g. waving, jumping), we attempt to identify simple gestures that a person might perform with his or her hand (e.g. grasping, pointing). To do this, we use a full body tracker developed by Hendrik Dahlkamp[2] and Christian Plagemann[3] to first identify the location of a person's hand (which should be positioned at the end of an arm in the inferred skeletal structure), and from there, recognize patterns in the hand's movement over time.

Pattern recognition consists of 1) identifying the pixels in the image that constitute the hand we're interested in, 2) extracting features from those identified pixels in order to classify the hand into one of a set of predefined poses, and 3) recognizing the occurrence of specific pose sequences as gestures. In the following sections we detail our approach in each of these areas, along with prior related work.

II. Identification of 'Hand Pixels'[4]

A. Prior work

While a full body tracker will give us a rough estimate of the hand's location, it remains for us to differentiate between pixels that are part of the hand ('hand pixels') and pixels that are part of the background ('background pixels'). One way to do this is to threshold based on depth. This involves estimating the depth value of the hand using our skeletal structure, and labeling those pixels whose z-value (depth) deviates too far from this estimated depth as background pixels. While this works well for more expensive cameras with high spatial and depth resolution (images with dimensions on the order of 100s to 1000s of pixels, and depth resolution on the order of a few millimeters [4]), the Kinect camera contains lower quality components. For a person standing approximately 3 meters away from the Kinect, the hand occupies a region of no more than 64x64 pixels. Furthermore, the resulting depth images suffer from IR occlusion and other effects; it is not surprising, therefore, that depth thresholding often results in a poor labeling (see Figure 1(a)).

Another approach is to label hand pixels using rgb data. One way to do this is to have a person wear a single colored glove, and label those pixels whose rgb values are close to that color as hand pixels [5]. A less robust method that does not require the use of a glove labels pixels based on their likeness to common skin colors [6].

B. Identification based on rgb data

Because we don't want to require the use of gloves, we use a skin detector for a preliminary estimation of our hand pixel locations. Jones, et al. [7] labeled the pixels in a set of 18,696 photographs (2 billion pixels) as skin or non-skin. They then fitted a Mixture of Gaussians model to the empirical data, resulting in an approximation for the quantities $p(RGB|S = 1)$ and $p(RGB|S = 0)$; RGB indicates a specific set of rgb values, and S is a binary random variable indicating whether or not a pixel is that of skin.

By looking at multiple samples of cropped images of our estimated hand regions, and labeling the pixels in the image as skin or non-skin, we can estimate $P(S)$. Using these probabilities, we can then calculate:

$$p(S|RGB) = \frac{p(RGB|S)p(S)}{\sum_{S'} p(RGB|S')p(S')} \quad (1)$$

This allows us to estimate, on a pixel-by-pixel basis, the probability that a pixel is a skin pixel.

C. Color balancing

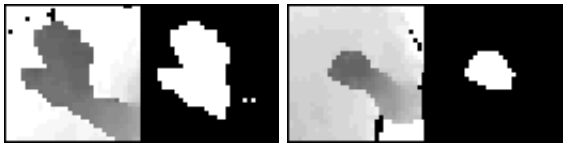
Most of the photographs used by Jones, et al. [7] to calculate $p(RGB|S)$ were taken in normal to bright lighting conditions, so our estimate of $p(S|RGB)$ is based on assumptions of similar lighting. This assumption, however, may not always

[1]The first two paragraphs in this section are taken directly from the introduction in [1], a paper I wrote last quarter.

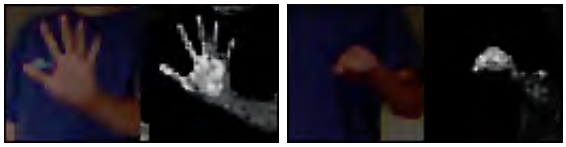
[2]Ph.D. Candidate in the Computer Science Department at Stanford University

[3]Post Doctoral Scholar in the Computer Science Department at Stanford University

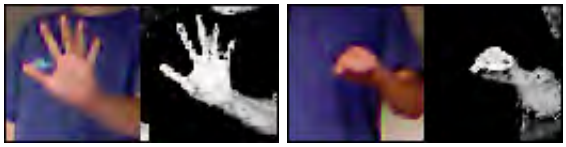
[4]Subsections A, B, and E paraphrased from [1], a paper I wrote last quarter.



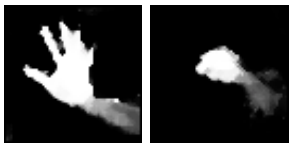
(a) Depth Thresholding



(b) Skin Estimate



(c) Skin Estimate with Color Balancing



(d) Combined Estimate

Fig. 1. Hand pixel estimates for an open and closed hand.

hold, and we see that our estimation of skin pixels may be inaccurate in poor lighting (see Figure 1(b)). To compensate, we color balance our pixel rgb values before calculating $p(S|RGB)$.

We find that in general, color balancing algorithms such as grayworld or scale-by-max do not work well because they sometimes drastically alter pixel r:g:b ratios, which in turn throws off our skin estimation. Instead, we propose a modified version of the scale-by-max algorithm that works well in practice.

For a pixel p with rgb values r_p, g_p, b_p, the original scale-by-max algorithm performs the following scaling:

$$r_p = \frac{r_p}{max(r)}; g_p = \frac{g_p}{max(g)}; b_p = \frac{b_p}{max(b)} \tag{2}$$

where $max(r)$ is the maximum value in the red channel, and similarly for $max(g)$ and $max(b)$. In our modified algorithm, we do the following:

$$r_p = \frac{r_p}{max(r,g,b)}; g_p = \frac{g_p}{max(r,g,b)}; b_p = \frac{b_p}{max(r,g,b)} \tag{3}$$

where $max(r,g,b) = max(max(r), max(g), max(b))$. Here, we don't scale the channels separately, and thus preserve the r:g:b ratios of all the pixels. Our assumption is that if at least one channel contains a value of 255, our image is bright enough for use with our skin detector. We see in Figure 1(c) that simple color balancing can make a big difference in our final skin estimate.

D. Exploiting spatial dependence

We notice in Figure 1(c) that there are multiple skin pixels with a low assigned skin probability that happen to be surrounded by skin pixels with a high assigned skin probability. It is generally the case that a pixel that is near multiple other skin pixels will have a higher probability of being skin, simply because skin is contiguous. To exploit this spatial dependence, we perform an image close operation (using a 3-pixel diameter cross structuring element) following our final skin estimate.

E. Integrating depth data

The Kinect sensor provides us with valuable, if coarse, depth data that we can use to further refine our hand pixel estimates. In particular, if we do a rough labeling of hand pixels for various sample images, we can estimate $p(D|H)$ where D represents a pixel's depth value, and H is a binary random variable indicating whether or not a pixel is that of a hand. We can then compute the following, assuming $(RGB, S \perp\!\!\!\perp D)|H$:

$$p(H, RGB, D) = \sum_S p(H)p(D|H)p(RGB|S)p(S|H) \tag{4}$$

$$p(H|RGB, D) = \frac{p(H, RGB, D)}{\sum_{H'} p(H', RGB, D)} \tag{5}$$

For simplicity, we assume that $p(S = H) = 1$, and therefore $p(H) = p(S)$. This is reasonable, since most hand pixels will be skin pixels, and vice versa.

Because it is extremely tedious to label hand pixels manually, our estimates of $p(H)$ and $p(D|H)$ come from the labelings obtained by $p(H|RGB, D)$; and this, in turn is calculated from $p(H)$ and $p(D|H)$. We therefore find $p(H)$ and $p(D|H)$ by using the Expectation Maximization (EM) algorithm, and this process, along with the resulting converged values of $p(H)$ and $p(D|H)$, are detailed in work I did last quarter [1].

We find that although our skin detector picks up non-hand skin pixels in Figure 1(c) (e.g. the neck region, skin-colored wallpaper), our final estimate that integrates depth data in 1(d) manages to eliminate such regions from our labeling. We assume $S = H$ in our calculations, but it is rarely the case that skin pixels occur at the depth of the neck or the background wall. So our depth information eliminates these false positives while reinforcing our original estimates of skin pixels.

We also notice that because the depth information is sometimes unreliable near the edges of a hand's fingertips, the introduction of depth data may introduce inaccuracies. This is an acceptable loss, because in instances where the rgb image is noisy (e.g. bad lighting) or the resulting skin estimate is inaccurate (e.g. skin-colored background), the integration of depth information greatly improves the fidelity of our final estimate.

III. Feature Extraction

A. Prior work

Given a labeling of hand pixels, we now want to identify it as one of a set of predefined poses. To do so, we extract a set of features that will allow us to differentiate between labelings.

Previous work was done to classify a hand into one of a set of signed words (sign language) [5], [6]. To do this,

blob descriptors (major/minor axis length ratio, eccentricity, orientation), radial histograms, and raw shape data were used. Such descriptors are also used in more general contexts [8].

Using these descriptors, a fixed background and orientation, a wrist marker, and high resolution images, an accuracy of 99.1% was achieved on a dictionary of 46 words in [6]. Using blob descriptors and lower resolution images, an accuracy of 86.5% was achieved on a dictionary of about 5 words in [8]. Our task is closer to the one found in [8] since we are using low resolution images with varying backgrounds, no wrist marker, and a small dictionary.

In the following sections, we will explore the use of some of the features described above, as well as a set of SURF [9] inspired features we developed ourselves.

B. Scale and Rotation Invariance

Before extracting our features, we want our images to be both scale and rotation invariant. We assume that a person is standing approximately 3 meters away from the Kinect sensor, providing imperfect but acceptable scale invariance. Traditional methods for rotation invariance (e.g. calculating the dominant gradient direction), however, do not work as well. This is because our images are low resolution and susceptible to noise. Furthermore, given an image of an open hand, the dominant gradient direction is often skewed by the relative orientations of the fingers.

Using the full body tracker, we can project the forearm corresponding to our hand of interest from the 3D model to our image perspective. We can then calculate the angle at which the arm is rotated, and rotate the image to some normalized angle. Doing this drastically improves the performance of our pose discrimination algorithms, and provides a more intuitive and consistent rotation normalization than the results of typical dominant gradient direction calculations.

C. Radial histogram

Because our cropped images are not perfectly centered, we must find the center of mass in our hand pixel estimate before we can compute the radial histogram described in [6]. Let $p(x, y)$ be the probability that the pixel at (x, y) is a hand pixel. Then the center of mass (x_c, y_c) is simply calculated from the following:

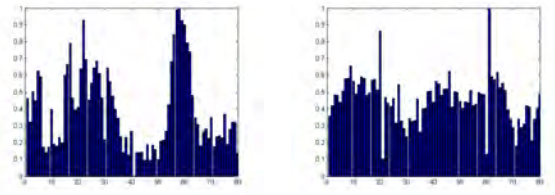
$$x_c = \frac{1}{N}\sum_{(x,y)} xp(x,y); y_c = \frac{1}{N}\sum_{(x,y)} yp(x,y) \tag{6}$$

where N is the total number of pixels.

Then for each pixel, we calculate the angle offset from this center. That is, for a pixel at (x, y), we calculate its angle offset:

$$a(x,y) = tan^{-1}(\frac{x - x_c}{y - y_c}) \tag{7}$$

Afterwards, we generate a weighted histogram of these offsets, binning $a(x, y)$ with weights $p(x, y)$, and normalize the resulting histogram so that the max binned value is 1. The idea here is that for any hand image, the corresponding radial histogram will have distinct spikes corresponding to extended fingers.



(a) Open Hand (b) Closed Hand

Fig. 2. Radial histograms for an open and closed hand using 80 equally spaced bins.

In Figure 2 we see histograms for an open and closed hand. As expected, the open hand has a histogram with more spikes and overall variation, whereas the closed hand has a more uniform histogram. The problem with this approach, however, is that because we are using low resolution images, the radial histograms are extremely susceptible to noise. Even a slight shift in our center of mass drastically alters these histograms, and as we'll see later, we are unable to reproduce the results in [6].

D. Modified SURF

The features described in [9] (SURF descriptors) are usually used for keypoint matching. Its use generally involves the acquisition of multiple keypoints in an image, followed by the computation of SURF descriptors for each of these keypoints. Because our cropped images are so small, however, we can treat the entire image as a keypoint.

We developed a set of features inspired by SURF. To calculate these features, we divide up our 64x64 cropped images into 64 8x8 subregions. Then for each region, we calculate the following:

$$\sum_{(x,y)} dx(x,y), \sum_{(x,y)} |dx(x,y)|, \sum_{(x,y)} dy(x,y), \sum_{(x,y)} |dy(x,y)| \tag{8}$$

where

$$dx(x,y) = \frac{p(x+1) - p(x-1)}{2}, dy(x,y) = \frac{p(y+1) - p(y-1)}{2} \tag{9}$$

Our final feature vector is simply the concatenation of these numbers, and therefore has a length of 256.

E. Performance

To test the performance of the features described above, we capture a sequence of 2901 hand images in canonical open and closed poses. We then train and test on these images and their corresponding features using a Support Vector Machine (SVM) with a 3rd order radial basis function. The results of 70-30 cross-validation are shown in table I.

For high resolution images, using a radial histogram will generally give better performance than raw pixel estimates [6]. But because the number of pixels we are dealing with is relatively small (4096), the corresponding feature vectors and the space they reside in are more compact. This leads to a reasonable separation of open and closed hand images, and better performance than obtainable by the radial histogram.

Feature Set	Correct Open	Correct Closed	Total Accuracy
Radial Histogram	178/399 (44.61%)	389/473 (82.24%)	567/872 (65.02%)
Raw Pixel Estimates	285/399 (71.43%)	448/473 (94.71%)	733/872 (84.06%)
Modified SURF	370/399 (92.73%)	468/473 (98.94%)	838/872 (96.10%)

TABLE I
RESULTS OF 70-30 CROSS VALIDATION ON A SET OF 2901 OPEN AND CLOSED HAND IMAGES (1327 OPEN, 1574 CLOSED).

We also see that using our SURF inspired features results in a better classification than obtained by both the radial histogram and the raw pixel estimates. This is probably because the original SURF descriptors were designed for small image regions, and that is exactly what we are dealing with here.

IV. GESTURE RECOGNITION[5]

Our method of gesture recognition simply identifies a sequence of poses. For example, an open hand followed by a closed hand can be labeled a 'grasping' gesture, while a closed hand followed by an open hand can be labeled a 'dropping' gesture.

This seems simple enough, but we can do better by exploiting the temporal dependence between poses. For example, if at time t the hand is in an open state, at time $t+1$ it will have a higher probability of also being in an open state rather than transitioning to a closed state. Let E_t be the estimated pose at time t, and let S_t be the actual pose (or state) at time t. Then using forward recursion on a simple Hidden Markov Model [10] (described in greater detail in work I did last quarter [1]), we can calculate:

$$p(S_t|E^t) = \frac{p(S_t|E^{t-1})p(E_t|S_t)}{\sum_{S'_t} p(S'_t|E^{t-1})p(E_t|S'_t)} \tag{10}$$

$$p(S_{t+1}|E^t) = \sum_{S_t} p(S_t|E^t)p(S_{t+1}|S_t) \tag{11}$$

where $E^t = \{E_1, E_2, \ldots, E_t\}$. We obtain $p(E_t|S_t)$ from the probabilities output by the SVM (calculated by fitting the decision value to a sigmoid curve). We estimate $p(S_{t+1}|S_t)$ and $p(S_0)$ manually, and use $p(S_t|E^t)$ as our final estimate of the hand pose at time t.

To demonstrate this, we use the example of a 'grasping' and 'dropping' gestures described previously. We have two states, 'open' and 'closed' which we'll label with '1' and '0' respectively. We set $p(S_0 = 1) = p(S_0 = 0) = 0.5$. And we set transition probabilities $p(S_{t+1} \neq S_t)$ = 0.1. The results of our initial SVM output (using our SURF inspired features), and the corresponding output after forward recursion, are shown in Figure 3.

We see that because we assume a low transition probability, forward recursion manages to suppress the noisy transitions we observe in the initial frame-by-frame estimates. Our initial mean squared deviation from the human labeling is 0.0945, and with forward recursion we succeed in reducing this to 0.0907.

[5]This section based on work done in [1], a paper I wrote last quarter.

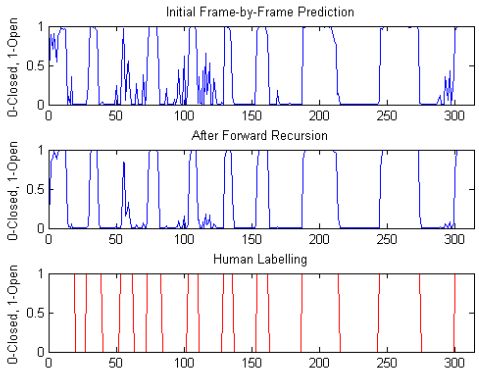


Fig. 3. Estimation on a captured image sequence (30 fps) of 'grasp' and 'drop' gestures.

V. CONCLUSION

In this paper, we proposed a method to integrate the rgb and depth information returned by the Kinect to provide a rough labeling of the hand pixels in a scene. This involves cleaning up the rgb image using standard image processing techniques like color balancing and dilation/erosion, and incorporating the depth data using a simple probabilistic model.

We then normalized the rotation on the resulting estimates, and extracted three sets of features (a radial histogram, the raw pixel estimates, and a modified version of the SURF descriptors). We compared the performance of these features on a set of 2901 open and closed hand images, and found that our version of SURF descriptors was best suited to the cropped hand images we were dealing with.

Finally, we demonstrated a simple method of gesture recognition, using forward recursion to make our results more robust to classification noise.

Using OpenCV, we wrote a program that does all of the above, allowing a user to interact with a virtual environment, grasping and dropping virtual objects simply be performing gestures in front of a Kinect sensor. This was demonstated in a presentation on June 6, 2011. Figure 4 shows a screenshot of our demo.

VI. ACKNOWLEDGEMENTS

I want to thank Hendrik Dahlkamp and Christian Plagemann for providing me with a Microsoft Kinect sensor, software to obtain cropped hand images, and access to the full-body tracker they developed. I want to thank Hendrik, especially, for helping me integrate my gesture recognition code with the full-body tracker, and for his help at the presentation. I'd also like to thank Zhi Li and Ngai-Man Cheung for some very

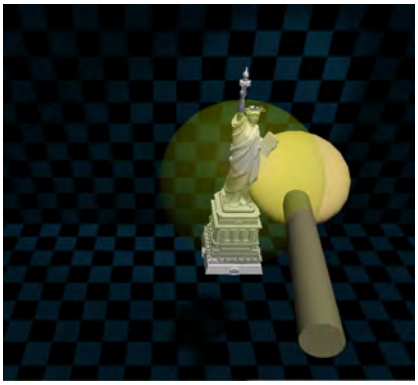


Fig. 4. Screenshot of the demonstrated program that allows you to drag and drop virtual objects.

helpful conversations, and Professor Bernd Girod for a great class that taught me the image processing techniques I used in this paper.

VII. Appendix

The Matlab files that were used to perform the relevant image processing techniques, calculate features, train and test on the cropped images, and generate the relevant plots are attached and written by me. I also wrote the code for a .dll file that contains definitions for two functions: 1) an init function that loads probability parameters and the trained SVM model (these files would be created and saved using the Matlab code), and 2) an update function that takes an rgb and depth image as its parameters, and outputs the probability that it is an open hand using the algorithm we described in this paper. This code has also been attached. The .dll was integrated into a program developed by Hendrik Dahlkamp and Christian Plagemann, and the result was demonstrated on June 6, 2011. A screenshot of this program is shown in Figure 4.

References

[1] M. Tang. "Hand Gesture Recognition Using Microsoft's Kinect." *Paper written for CS228*, Winter 2010.

[2] R. Urtasun and P. Fua. "3D Human Body Tracking Using Deterministic Temporal Motion Models." *Lecture Notes in Computer Science*, vol. 3023, 2004: 92-106.

[3] J. Sullivan and S. Carlsson. "Recognizing and Tracking Human Action." *Lecture Notes in Computer Science*, vol. 2350, 2002: 629-644.

[4] X. Zabulis, H. Baltzakis and A. Argyros. "Vision-based Hand Gesture Recognition for Human-Computer Interaction." *Computer Vision Techniques for Hand Gesture Recognition*.

[5] A. Zafrulla, H. Brashear, H. Hamilton, T. Starner. "A novel approach to American Sign Language (ASL) Phrase Verification using Reversed Signing." *Computer Vision and Pattern Recognition Workshops*, 2010.

[6] R. Lockton. "Hand Gesture Recognition Using Computer Vision." http://research.microsoft.com/en-us/um/people/awf/bmvc02/project.pdf.

[7] M. J. Jones and J. M. Rehg. "Statistical Color Models with Application to Skin Detection." *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1999.

[8] L. Bretzner, I. Laptev, T Lindeberg. "Hand Gesture Recognition using Multi-Scale Colour Features, Hierarchical Models and Partical Filtering." *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, 2002.

[9] H. Bay, T. Tuytelaars, L. Van Gool. "SURF: Speeded Up Robust Features," *Lecture Notes in Computer Science*, 2008.

[10] L. E. Baum. "An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes." *Inequalities*, vol. 3, 1972:1-8.

Hand Gesture Recognition Using Microsoft's Kinect

Matthew Tang

March 16, 2011

1 Introduction

The advent of relatively cheap image and depth sensors has spurred research in the field of object tracking and gesture recognition. One of the more popular devices used to do this type of research is the Microsoft Kinect, which has sensors that capture both rgb and depth data. Using similar data, researchers have developed algorithms to not only identify humans in a scene, but perform full body tracking; they can infer a person's skeletal structure in realtime, allowing for the recognition and classification of a set of actions (e.g. [1], [2]).

In this paper, we use data collected from a Microsoft Kinect to explore the feasibility of gesture recognition on a micro scale. Instead of attempting to recognize full body actions, we attempt to identify simple gestures that a person might perform with his or her hand (e.g. grasping or pointing). To do this, we use the full body tracker developed by Hendrik Dahlkamp[1] and Christian Plagemann[2] to first identify the location of a person's hand (which should be positioned at the end of an arm in the inferred skeletal structure), and crop out an image of where we think the hand should be. For a person standing 3 meters away from the sensor, the result is a 64x64 rgbz image. We capture a sequence of such images, and our goal is to detect the occurrences of gestures as a person performs them.

2 Traditional Methods

Before we can perform gesture recognition, we need to identify roughly where the hand is located in our images. The most common way to do this is to threshold based on depth. This involves inferring a rough depth using the skeletal structure, and cropping out those pixels whose z-value (depth) deviates too far from this estimated depth. While this works well for more expensive cameras with high spatial and depth resolution (images with dimensions on the order of 100s to 1000s of pixels, resolution on the order of a few millimeters [3]), we are using a much cheaper Kinect. The images we are dealing with have dimensions on the order of 10s of pixels, and although our depth has a nominal accuracy of 3mm, the depth images provided by the sensors succumb to IR occlusion and other effects (see Figure 1(b)). In our paper, we explore the use of rgb data, in addition to depth data, to enhance our estimates of hand locations.

Once we have a basic outline of our image, a common approach to gesture recognition is to perform fingertip detection (using curvature measures [3]) and tracking. We find, however, that our images have too low a resolution for such curvature methods to work well. But because our images have such a low resolution, we find that using an SVM with our estimated hand location as a feature vector works better than it would for large images (where the pixel space

[1]Ph.D. Candidate in the Computer Science Department at Stanford University
[2]Post Doctoral Scholar in the Computer Science Department at Stanford University

(a) RGB (b) Depth (c) Threshold

Figure 1: Sample data and results using only depth thresholding.

would be much larger). In addition to using SVMs, we extend this estimation scheme with a Hidden Markov Model, to produce a fairly robust gesture recognition algorithm.

3 Our Implementation

Our task can be broken up into two parts, 1) locating the hand and 2) classifying gestures based on temporal changes in hand location. We estimate the location of the hand by taking the following steps:

1. Assign to every pixel a prior probability of it being part of a hand.
2. Aggregate these pixels into superpixels (SPs), each with its own prior.
3. Define a cluster graph with every two adjacent SPs corresponding to a pairwise clique, and perform inference on this cluster graph.
4. Threshold on the resulting inferred probabilities.

After performing the above steps, we should have a hand location estimate that is better than what we see in Figure 1(c). Now, we look at sequences of such estimates to identify gesture sequences by doing the following[3]:

1. Identify a sequence of hand positions that comprise a gesture (e.g. an open hand followed by a closed hand might comprise a grasping gesture).
2. Use a Support Vector Machine (SVM) to differentiate between these hand positions, using our inferred hand location as a feature.
3. Estimate the result using a Hidden Markov Model and Forward Recursion.

We will now discuss each of these steps in detail.

3.1 Estimating Pixel Probabilities

We can only estimate pixel probabilities based on rgbz values. It is reasonable to assume that color and depth information are independent given the knowledge of whether a pixel is part of a hand, so we consider the rgb and z values separately (given H, a binary variable indicating whether or not a pixel is part of a hand). Furthermore, the data and methods in [4] provide a way for us to obtain rgb distributions on skin and non-skin objects. By sampling rgb values from a vast sample of skin and non-skin images, we can construct two 256x256x256 probability matrices $p(RGB|S = 1)$ and $p(RGB|S = 0)$, where S is a binary random variable indicating whether or not a pixel is that of skin. This should be extremely helpful since there is a strong correlation between the presence of a hand and the presence of skin. Finally, using our intuition

[3]Based partially on work I completed in CS 229 (Machine Learning), where a higher resolution camera was used.

about the relationship between our data, our available probability information, and our goal of labeling pixels as belonging or not belonging to a hand, we decide to use the probabilistic graphical model depicted in Figure 2 to describe each pixel.

We have a reasonable way to approximate $p(RGB|S)$, but we have yet to determine the parameter $\theta_H = p(H = 1)$, and similarly, the parameters $\theta_{D|h^1}$, $\theta_{D|h^0}$, $\theta_{S|h^1}$, and $\theta_{S|h^0}$. So we start by assigning reasonable initial values to these parameters. By eyeballing the image in Figure 1(a) and other images obtained from our Kinect sensor, we assign a rough estimate $\theta_H = 0.4$. Given that our hand is estimated to have a depth value of 127 (with depth values ranging from 0 to 255), we assign $\theta_{D|h^1}$ to take on values from discretized normal distribution centered at $D = 127$ with spikes in probability at $D = 0$ and $D = 255$ to account for the IR occlusion effects we see in Figure 1(b). The result is a normalized vector of length 255. For simplicity, we let $\theta_{D|h^0} \sim Unif[0, 255]$. We make the assumption that the hand will almost always be made up of skin, so we set $\theta_{S|h^1} = 0.99$.[4] Finally, we set $\theta_{S|h^0} = 0.95$, accounting for the fact that there may be some non-hand skin pixels in our frame (e.g. if the hand were held close to the face).

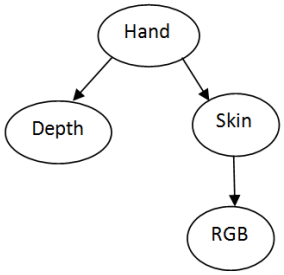


Figure 2: Our Pixel-Level Bayesian Network

Using these parameters, we can now perform variable elimination on our Bayesian Network given the evidence set $\{rgb, d\}$. More specifically, we compute the following:

$$p(H, rgb, d) = \sum_S p(H)p(d|H)p(rgb|S)p(S|H) \tag{1}$$

$$p(H|rgb, d) = \frac{p(H, rgb, d)}{\sum_H p(H, rgb, d)} \tag{2}$$

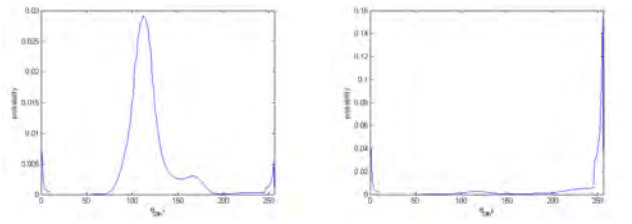
For every pixel, we assign $p(H|rgb, d)$ as the 'prior probability' that the pixel is part of a hand. This allows us to proceed to steps 2-4 in our hand location method. Afterwards, we will have an initial estimate of the location or our hand. And using this estimate, we can further refine our parameters.

We now have the observations $h[m]$, $rgb[m]$, $d[m]$, and missing data $s[m]$. We can immediately set $\theta_H = \frac{M[h^1]}{M}$ and $\theta_{D|h^i} = \frac{M[D,h^i]}{M[h^i]}$, where $M[x]$ represents a count of the number of occurrences of x in our dataset and M is the total number of elements in our dataset. We can also infer soft counts of $M[S, H] = \sum_m p(S, H|h[m], rgb[m], \theta)$ to estimate $\theta_{S|h^i} = \frac{M[S,h^i]}{M[h^i]}$. We then iterate, performing steps 1-4 again and updating our parameters in a similar fashion to Expectation-Maximization (EM). Note that this is not guaranteed to converge, since our dataset changes on every iteration. Ideally, we would obtain our dataset by labeling the images by hand, but this is hard to do since our training set consists of a few thousand 64x64 images.

In practice, for an arbitrary initialization of parameters, our modified EM algorithm converges to values that produce very inaccurate results. If our initial set of parameters results in a poor estimate, this estimate will give us an innacurate 'dataset', culminating in suboptimal updates to our parameters. Such errors tend to propagate, so that for more iterations of this algorithm, our estimates will become more unreliable (e.g. oftentimes it will result in an estimator that never reports a pixel as being part of a hand).

To overcome this, we ran our algorithm with the initial set of parameters described above. We then handpicked estimates that seemed to match up with the observed reality, making

[4]We avoid assigning probabilities of 0 or 1. Note that we can tweak this in the future to account for the presence of gloves.



(a) Depth Given Hand (b) Depth Given Not Hand

Figure 3: Estimated Parameters.

sure to include estimates from a diverse set of images (since we don't want a biased sample). We used these estimates as our dataset, and inferred new parameters based on these counts. Finally, because the domain of D is large and our dataset may not have sufficiently saturated it, we smoothed $\theta_{D|h^1}$ and $\theta_{D|h^0}$ using a moving average window. We repeat this process for a few iterations, and we find that we are able to obtain marked improvements in our estimator. This assessment is qualititative since we don't have a 'correct' dataset to compare with.

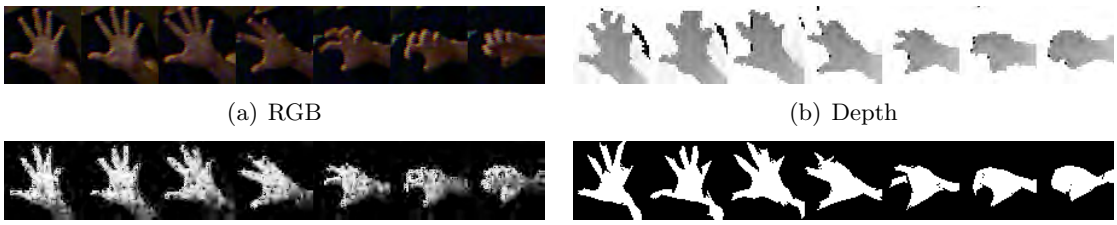
We find that parameter values that produce good results are $\theta_D \approx 0.22$, $\theta_{S|h^1} \approx \theta_{S|h^0} > 0.99$ (which suggests that the variable S may not be necessary for our usage patterns), and $\theta_{D|h^1}, \theta_{D|h^0}$ shown in Figure 3. Pixels that are part of the hand tend to have a depth near 127, with the occasional pixel having an extremely large or small depth due to IR occlusion. Pixels that are not part of the hand tend to be in the background, with a somewhat noticeable number of pixels with a depth near 127 due to the smearing of depth values between fingers (see Figure 1(b)).

3.2 Superpixels and Inter-Pixel Factors

So far, we have only considered our images on a pixel by pixel basis. Now, we focus on improving our estimates by exploiting the relationships between pixels. Before we do anything else, we segment our image into superpixels. This provides two distinct benefits. First, by using SPs, we reduce the complexity of any cluster graph that we might consider using to describe the relationship between adjacent image elements. And second, SPs naturally hug object outlines, allowing for a labeling that gives a tight estimate on our hand location that might otherwise be hard to achieve.

We partition our image using the Normalized Cuts method described in [5], adapting the code referenced in the paper. Our goal now is to infer the value of a binary variable H for every SP, where H indicates whether or not that SP is part of a hand. We define a set of factors as follows. There is a singleton factor for every SP. To compute the value of the factor $\phi_X(H_X)$ for some SP X, we average the 'priors' of every pixel a contained within X, *i.e.* $\phi_X(H_X) = \frac{1}{|X|}\sum_{a \in X} p_a(H|rgb_a, d_a)$ where $|X|$ is the number of pixels in X. We also define pairwise factors between every two adjacent SPs. For two adjacent SPs X and Y, the corresponding factor takes on the value $\phi_{XY}(H_X, H_Y) = w_1$ if $H_X = H_Y$, w_0 otherwise. The ratio $\frac{w_1}{w_0}$ can be seen to describe the affinity for adjacent superpixels to take on the same value. When this ratio is 1, the pairwise factors are rendered inactive.

Finally, we construct a cluster graph from these factors, and perform approximate inference. This yields our final estimate for our hand location. We threshold the resulting probabilities, whereby an SP is declared part of the hand if its probability $p(H = 1)$ post-inference is greater than some threshold T. Normally, we would set $T = 0.5$, but we find that we get more visually



(a) RGB (b) Depth

(c) Pixel Prior (d) Final Estimate

Figure 4: Our algorithm performed on an image sequence.

accurate results if we set this threshold to be around $T = 0.4$. Figure 4 shows the result of running our algorithm on a sequence of images. We immediately see that our algorithm produces cleaner results than that achieved simply by depth thresholding (e.g. Figure 1(c)).

We find that if we choose a lower ratio $\frac{w_1}{w_0}$, our estimator performs better on open hands, and if we choose a higher ratio, our estimator performs better on closed hands. This is because for an open hand, many of the SPs that are part of the hand are adjacent to SPs that aren't part of the hand, especially along the fingers. For a closed hand, the SPs that define the hand are generally clumped together and adjacent to other SPs that take on the same value. In the next section, we use an SVM to classify a hand as being 'open' or 'closed', and we use a hidden markov model to both estimate the current hand state, and predict the next hand state. Using this prediction, we can dynamically adjust the ratio $\frac{w_1}{w_0}$; the more confident we are that the next state will be an open hand, the lower the ratio we choose to use.

3.3 Learning Hand Positions and Classifying Gesture Sequences

We now want to be able identify a sequence of images with a particular gesture. To demonstrate the feasibility of doing this, we will focus on one specific gesture: the 'grasping' gesture (see Figure 4(a)). For simplicity, we will define such a gesture as an 'open' hand, followed quickly by a 'closed hand'.

First, we want to be able to classify a single image as an 'open' or 'closed' hand. We do this by labeling hand images in a training set as either 'open' or 'closed'. We then take our corresponding estimates of hand location (see Figure 4(d)) and feed this into an SVM using a 3rd order radial basis function[5]. Afterwards, we validate on a test set. In our case, we used 70-30 cross validation and were able to obtain an accuracy of $\sim 80\%$.

Next, we assume an underlying hidden state S_t for the hand at every timestep t. This state is binary, taking the values of 'open' or 'closed'; it influences the rgbz image we observe, and through it, our location estimates and the output of our SVM. We define a variable O_t to represent our observations, and we let $p(O_t|S_t)$ be the confidence with which the SVM classifies the current image as being in state S_t. We also define a relationship between S_t and S_{t+1}, where $p(S_{t+1}|S_t)$ is the probability of the next state given the current state. Our intuition tells us that the next state should be equal to the previous state with high probability, so we set this parameter accordingly. Our resulting temporal model is shown in Figure 5.

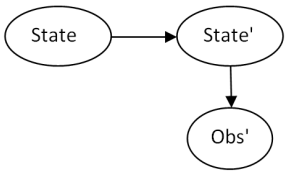


Figure 5: Our Temporal Bayesian Network

[5]Used the SVM library created by Chih-Chung Chang and Chih-Jen Lin, which is available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm/.

Now, we can use simple message passing to determine $p(S_t|O^t)$, where $O^t = \{O_1, O_2, \ldots, O_t\}$. For the temporal Bayesian Network we defined, this is also referred to as forward recursion [6]. In addition, we can predict the next state $p(S_{t+1}|O^t)$, allowing us to dynamically adjust the weights w_1 and w_0 in our SP cluster graph mentioned in the previous section.

Using the prior $p(S_t) = 0.5$ and setting $p(S_{t+1} = S_t) = 0.9$, we perform forward recursion on a test sequence of five grasping motions, and the result is shown in Figure 6. While forward recursion doesn't significantly improve our accuracy, it improves the robustness and confidence of our estimates. We also note using this method, we are able to recognize a grasping gesture with near 100% accuracy.

4 Conclusion

In this paper, we proposed a method for hand gesture recognition using Microsoft's Kinect. Our approach involved the use of a pixel-level Bayesian Network, superpixels, a cluster graph created from these superpixels, an SVM, and a separate temporal Bayesian Network. We found that the most important aspect of our algorithm was the pixel-level Bayesian Network. And the key to getting this part of the algorithm to work was choosing the right parameters. Our initial parameters yielded results that were noisy, and although they didn't look bad they resulted in unreliable singleton factors in the cluster graph portion of our algorithm. By carefully tuning and updating the parameters, we were able to get our parameters to converge (see Figure 3), and obtain 'priors' that matched up extremely well with reality (see Figure 4(c)). In fact, the priors worked well enough so that inference on the SP cluster graph we defined earlier did little to improve our results. In addition, the parameters we chose for this Bayesian Network suggested that a differentiation between skin and hand may be unnecessary.

The SVM we used allowed us to quantify the improvements of our hand location algorithm. Running the SVM on depth thresholded estimates yielded an accuracy of $\sim 74\%$, which we were able to bump up to $\sim 80\%$ with our hand location algorithm. This improvement can also be seen qualitatively in Figure 4(d). Finally, we found that using a simple temporal model can greatly improve the robustness of our system to chance misclassifications by the SVM (see Figure 6).

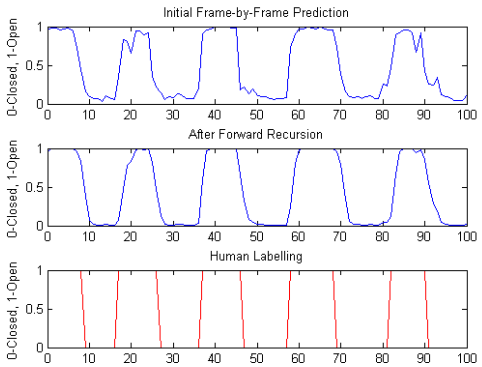


Figure 6: Estimation on a captured image sequence.

5 Acknowledgements

I would like to thank David Knight for pointing me to [4] and providing me with rgb data on skin tones. I'd also like to thank Hendrik Dahlkamp and Christian Plagemann for providing me with advice, a Microsoft Kinect, and software to obtained cropped hand images. Finally, I'd like to thank Professor Daphne Koller and the CS 228 course staff for a great course, and for their helpfulness in answering my questions.

References

[1] R. Urtasun and P. Fua. "3D Human Body Tracking Using Deterministic Temporal Motion Models." *Lecture Notes in Computer Science*, vol. 3023, 2004: 92-106.

[2] J. Sullivan and S. Carlsson. "Recognizing and Tracking Human Action." *Lecture Notes in Computer Science*, vol. 2350, 2002: 629-644.

[3] X. Zabulis, H. Baltzakis and A. Argyros. "Vision-based Hand Gesture Recognition for Human-Computer Interaction." *Computer Vision Techniques for Hand Gesture Recognition.*

[4] M. J. Jones and J. M. Rehg. "Statistical Color Models with Application to Skin Detection." *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1999.

[5] T. Cour, F. Benezit and J. Shi. "Spectral Segmentation with Multiscale Graph Decomposition." *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[6] L. E. Baum, "An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes." *Inequalities*, vol. 3, 1972: 1-8.