Classifying American Sign Language Letters

CSE 5524

Dr. Jim Davis

By: Mark Beshara, Jeremy Sandrof, Pranav Suresh, James Enouen

Introduction

Many people with impairments that affect speech and hearing rely on Amerian Sign Language (ASL) for face-to-face communication. However, the majority of the general public does not understand ASL, which results in a communication gap between ASL speakers and non-ASL speakers. This is undesirable as it leads to inefficient communication and generally inconveniences both parties involved. Therefore it is important to have solutions which bridge this gap so that non-ASL speakers are able to communicate with ASL speakers. This project explores this problem by examining whether ASL letters can be interpreted to English letters using basic computer vision techniques. To further constrain the problem, only ASL letters which do not involve motion will not be considered, and only a subset of the remaining letters will be considered. The letters that will be classified are V, W, L, Y, B, U, and H. This set of letters was selected since it provides diversity in letter shape, but there also exist similarities between letters in this set. The ASL signs associated with each letter can be seen in figure 1.

To accomplish this task, the effectiveness of template matching and similitude moment matching were examined. These techniques were selected since moments provide a generalizable solution due to their ability to describe the region shape of an image, which reduces the granularity of the information but maintains important information like structure. However, from observing image 1, several signs such as V and U will have similar region shapes, possibly resulting in misclassifications. Therefore template matching poses a good alternative to address such issues at it examines the image as a whole and can evaluate more minute differences in the image.

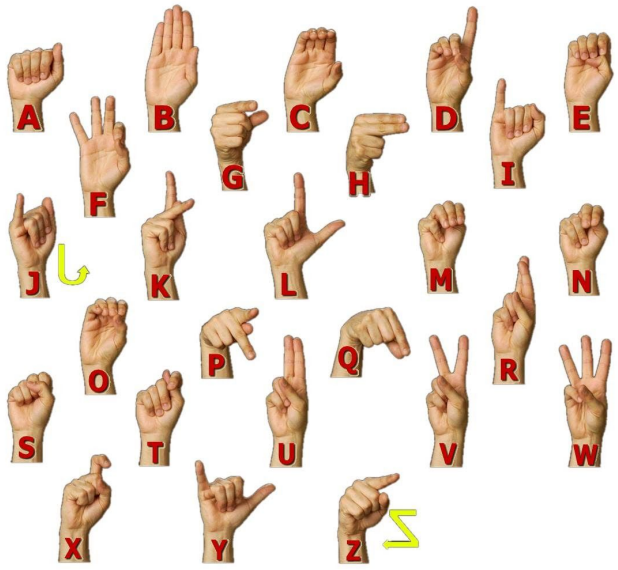

Figure 1: ASL letters

Techniques

1. **Similitude Moment Matching**

   The first technique considered for classifying ASL signs was to compute similitude moments for a binary hand shape in every frame captured by the Azure Kinect, and then comparing the computed to a vector of similitude moments for that frame to a set of template similitude moments. The template similitude moments were precomputed for various ASL signs, with each template being assigned to a particular letter in the English alphabet. The equation for computing similitude moments can be found in equation 1, and computing the moments produces a vector N of length seven, whose elements are the moments in the order specified in figure 2. Similitude moments were selected as the moments of choice to compare due to their invariance to scale and translation, as a sign could appear anywhere in the image and at varying distances which results in varying sizes.

$$\eta_{ij} = \frac{\mu_{ij}}{(m_{00})^{\frac{i+j}{2}+1}} = \frac{\sum\sum (x-\bar{x})^i (y-\bar{y})^j I[x,y]}{\left(\sum\sum I[x,y]\right)^{\frac{i+j}{2}+1}}$$

$$\text{for } 2 \le (i+j) \le 3:$$

Equation 1: Similitude Moments

$$\mathrm{N} = \begin{bmatrix} \eta_{02} & \eta_{03} & \eta_{11} & \eta_{12} & \eta_{20} & \eta_{21} & \eta_{30} \end{bmatrix}$$

Figure 2: Vector of length 7 produced by computing similitude moments

The template with the minimal distance to the similitude moments vector for any given frame was then selected as the best match, and the letter associated with that template was assigned to that frame as the classification of what english letter was being represented as an ASL letter in the image.

In order for this method to work, a binary image mask had to be constructed of the object thought to be the hand in each image. To accomplish this task, the depth image produced by the kinect was utilized, by first thresholding the image from the minimum value in the depth map to the minimum value plus some constant value, with all other values set to 0. This was done in order to isolate the hand from the rest of the image, since the natural motion for signing a letter in ASL is to extend the arm away from the body, making the hand the closest object in the depth map, and subsequently the object with the smallest values. The threshold uses two bounds with the constant value to account for the depth of

the hand. This value was experimentally determined through trial and error and was determined to be 110. After thresholding the image, the non-zero pixel values were scaled to 1 to produce a binary mask which captured the shape of the nearest object in the image which should be the hand in the shape of some ASL sign. Finally the binary mask was then opened by performing dilation and erosion to remove any noisy values that appear to protrude from the mask. This mask was then used to construct the similitude moments vector which would then be compared to the template vectors.

A visual representation of the depth map produced by the Azure Kinect can be found in figure 3, and the resulting binary mask produced by the process described above can be seen in figure 4.

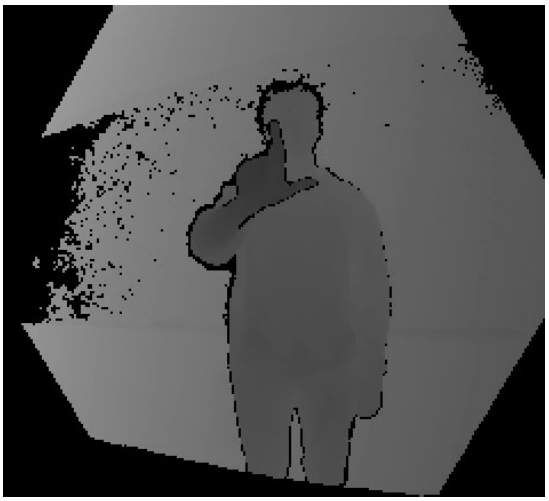

Figure 3: Depth map taken from Azure Kinect

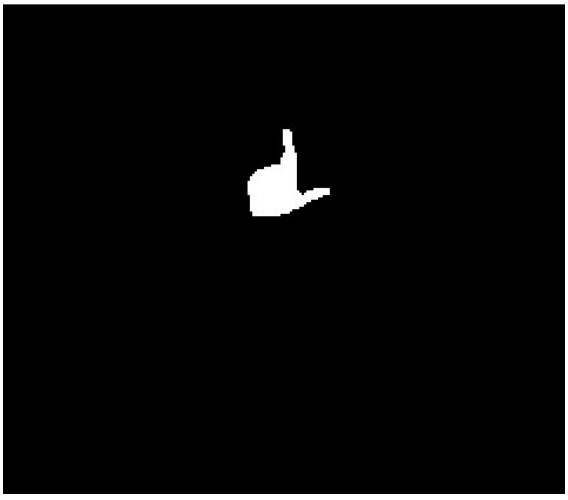

Figure 4: Binary mask produced from depth map

2. **Template Matching**

The second technique considered for classifying ASL signs was using template matching. Two forms of template matching were considered, sum-of-squared distances (SSD), and normalized cross-correlation (NCC). SSD was selected as it was computationally more efficient than NCC, however NCC provided more generalizability as it accounted for differences in brightness between images. The equations for computing SSD and NCC between a patch P and template T can be found in equations 2 and 3.

$$SSD(P,T) = \sum_{R,G,B} \sum_{x,y} (P(x,y) - T(x,y))^2$$

Equation 2: Equation for computing SSD between a patch and template

$$NCC(P,T) = \sum_{R,G,B} \frac{1}{n-1} \sum_{x,y} \frac{(P(x,y) - \bar{P}) \cdot (T(x,y) - \bar{T})}{\sigma_P \sigma_T}$$

Equation 3: Equation for computing NCC between a patch and template

Manually drawn templates were then created for each of the letters being classified by manually cropping a full RGB image from the Azure Kinect camera to the minimal patch around the hand and assigning it a label based on what letter appeared in the image. Each template would then be run against a frame from the camera and the highest NCC score or the lowest SSD score of all the templates was then considered to be the correct letter, and the letter associated with that template was assigned to the frame as the correct classification.

In searching the image using the templates, two differing techniques were considered. The first technique was to simply brute force the image scanning each possible patch in the image with each template. Due to the volume of templates (7 templates), and the large size of the images (1536x2048), this was not computationally feasible in real time. To increase computational performance, it was attempted to increase the stride from 1 to some larger value, so that a stride of 10 would correspond to 10 patches being skipped. While increasing computational speed, this is dangerous as it increases the likelihood of an incorrect classification occuring, as it is possible that the patch containing the hand in the full image was skipped completely, or only a fraction of the hand was in the image. The higher the stride, the higher the computational performance but at the cost of classification performance.

To avoid this issue, the other technique for matching templates was to find only the patch in the RGB image that corresponds to the hand found from the Depth Mask. This was done by finding the Binary hand mask, and finding the pixel locations of the hand on the depth map. Through experimentation of various approaches, we tried to find a correspondence between the points on the depth map and the RGB image. The RGB hand template was cropped out of the full RGB image and resized to a similar aspect ratio as our pre-computed templates. This would add scale invariance to our template matching technique, something that the brute force method did not compensate for. Then, we performed template matching method of classification between the pre-computed templates and our hand sign template extracted from the RGB image.

Since there was no point-to-point correspondence found between the RGB image and the depth mask, the templates extracted from the RGB image were not effective in the classification of sign language letters. Thus, we continued using the similitude moments technique for our demo.

Results

Method	Similitude Moments			Template Matching	
	Euclidean	Normalized Euclidean	Cosine Similarity	SSD w/ Stride 10	NCC w/ Stride 10
Accuracy	.71	.71	.76	.14	.13

To test the accuracy of each of our techniques, we accumulated a separate test set of each member of our team signing the seven letters which we performed the classification on. For the first technique using similitude moments, we achieved relatively good performance on the test set, remaining above 70% accuracy. In practice, it seemed that we only had trouble distinguishing the letters U, V, and W. Our first evaluation used Euclidean distance between the 7 dimensional similitude moment vectors. The next distance metric we used was we first normalized each of the 7 dimensions to have zero mean and unit covariance, usually referred to as ‘whitening’ or ‘normalizing.’ Afterwards, we computed Euclidean distance for these whitened 7D vectors. The final score we used for similitude moment matching was cosine similarity between the two vectors. For template matching, we dealt with many more issues which we were not able to overcome which lead to a much lower accuracy. We attribute the main issues to calculating the location of the hand in the RGB image given its position in the depth image as well as to accounting for the different skin colors in our test set using a single template. The brute force solution also did not account for scale invariance.

Problems

The initial problems encountered were due to the compatibility of the Azure Kinect and the programming language being used. All code for this project was developed in Python, however the Azure Kinect only supports C++/C. In order to accommodate the desired language, utility code was provided from the Azure Kinect Github repository that allowed for the creation of a c++ pipe server, and a corresponding python script which was capable of reading the images sent from the server. The c++ server and the python code then needed to be modified to fit the needs of the project.

Other problems encountered were related to finding a mapping between the RGB camera and the depth camera to be able to be utilized in template matching. From the documentation, with the RGB camera in 4:3 aspect ratio and the depth camera in Narrow field-of-view (NFOV), the field of view (FOV) would be similar considering both cameras, with the RGB camera having a larger FOV. The assumption was then we could simply map the depth image to the RGB image with a 1:1 mapping with the depth image centered on the RGB image. However, this mapping was not sufficient, and another suitable mapping could not be could not be found, resulting in template matching having to be done in a brute force manner, which removed the possibility of running it in real time.

A final issue we encountered is the computational bottlenecks for various algorithms. Running template matching on 7 templates, even with large strides, was still a monumental task and took up to a minute to compute a single frame, rendering it impossible to be used in real time which was a large goal of this project. Even with similarity matching, it took up to several seconds for a single computation to be done, which resulted in issues when trying to display the depth maps in real time. To account for this, a counter was implemented to skip frames so that the computations would not interfere with rendering the images on screen, however this produced a choppier stream, and was therefore not as aesthetically pleasing. The added time due to the computation also did not meet the project goals of having a truly real time system.

Discussion and Lessons Learned

The team learned many different lessons in completing this project. Primarily, we learned that depth maps are extremely powerful structures for analyzing a scene. However, they are quite susceptible to noise. For example, referring to figure 3 above, there is quite a lot of fuzziness on the left edge of the image. Also, the depth values near the edge of the person's silhouette are 0 (black noise near edges). Although this is a relatively easy fix, it goes to show that the IR technology used by the Azure Kinect to compute depth maps is not perfect.

Another lesson the team learned is to not engineer a solution that is too tightly tuned to specific parameters. In other words, when we were creating the project and taking images to use for

templates, parameters of the environment should have been paid attention to more. For example, the height of the camera above the ground, angle of the camera, distance to a wall in the background, and distance of the hand from the camera, to name a few. Changing one of some of these values would greatly affect the reference vectors being compared against. We learned this lesson when we tried to test our system with the camera at different heights/angles; hand signs would be classified incorrectly due to tiny discrepancies in distance scores

Future Work

Our future work can be broken up into a few primary goals: including the entire alphabet; testing on a much larger dataset with greater variance in hand shape and size; and achieving real-time computation.

Including the entire alphabet presents a few new challenges. Letters like J and Z required motion to sign, so we would need to add motion history to be able to classify these letters which would expand on our current motion detection. But this would also involve ensuring that the correct amount of frames are used in the motion history, so as to not interfere with the classification of other hand signs. Adding more letters would also require that we tightened the accuracy of our current algorithm because with a larger number of class labels, it is easier for the model to make misclassifications. In addition, it is likely we would need to fix our implementation of template matching to accommodate letters like M and N which are extremely similar in visual appearance and become nearly indistinguishable if only a silhouette is provided.

If we update our NCC method, we would need to overcome the associated challenges we mentioned with these methods: alignment between the RGB image and depth image and also the differences in skin tone between participants. Hopefully, using a much larger dataset and updating our methods to respect this larger space with extra templates or more modeling would overcome these challenges.

Finally, a main goal for this project if it were to continue is to create a real-time version of the software. Being able to compute this information in real-time allows the information to be communicated in real-time which brings us back to the original problem of identifying sign language is useful for removing communication barriers. If the communication is still delayed by 3 seconds, it is difficult to capture the full sentiment of the message.