A Visual Password Recognition Application Based On the American Sign Language

[1]Cory Shirts, [2]Kevin Ellsworth, [3]Wei Dang

Department of Electrical and Computer Engineering
Brigham Young University
Provo, Utah, 84604, USA

[1]c.shirts@gmail.com, [2]Kellsworth1010@gmail.com,
[3]weidang@et.byu.edu

***Abstract* – Many authentication systems have been developed based on various techniques, such as vocal, visual, and thermal. In this class project, we take an initial step in developing a visual password recognition system based on American Sign Language (ASL). Hand gesture recognition is not a new idea. In fact, there have already been more than a few implementations, some of which are in fact rather impressive. To distinguish us from the existing implementations and to fit in the time frame of a semester project, we aim to incorporate this hand gesture recognition technique into a practical application, where ASL is used as visual input to unlock the preset password. Current implementation supports password composed of 24 alphabetical letters and 10 numeric digits. We utilize template matching functions, provided in the OpenCV library, for hand gesture recognition which gives us good but not perfect results.**

Keywords – Visual authentication, American sign language recognition, marker, OpenCV

I. INTRODUCTION

This is a semester project, where we are required to develop a real time visual application. Our initial idea was to implement an obstacle avoidance algorithm on a robotic vehicle. However, this plan fell through due to various reasons.

Our backup plan is to develop a visual authentication application based on ASL using OpenCV library. This is inspired by the successful debut of Kinect, Microsoft's visual gaming gadget. Nonetheless, our application is much simpler, using only one ordinary webcam to recognize static hand gestures.

We take advantage of the existing template matching function implemented in the OpenCV library and use various tweaks to accelerate the image processing as well as to improve accuracy. The result is acceptable as long as the real time gestures resemble our templates. However, current implementation does not seem to work well if ambient setting becomes different from when the templates are taken.

II. SIMILAR WORK

Hand gesture recognition is not a new idea**.** It is nearly as old as the introduction of computer vision. Specifically, many approaches have been made using cameras and computer vision algorithms to interpret sign language. Just as speech recognition can transcribe speech to text, hand gesture recognition software is able to transcribe symbols represented through sign language into text [1].

The Media Lab at MIT has done an excellent job in implementing two Markov model-based systems for recognizing sentence-level continuous ASL using a single camera to track the user's unadorned hands, where they have achieved 92% to 98% word accuracy with both desk- mounted and cap-mounted camera [2].

Our work is not as ~~complicated~~ complex as the implementation at MIT's media lab. However, we apply this technique to practical life, making it a fun authentication application. In addition, we use existing OpenCV library functions rather than writing the application from scratch, which saves us a great deal of time.

III. OPENCV LIBRARY

OpenCV (Open Computer Vision Library) is a library of programming functions mainly aimed at real time computer vision, originally developed by Intel and now supported by Willow Garage [3]. The library is cross-platform.

Since version 2.0, OpenCV includes both its traditional C interface as well as a new C++ interface, which seeks to reduce the number of lines needed to code up vision functionality as well as reduce common programming errors such as memory leaks that can arise when using OpenCV in C. Our project uses OpenCV version 2.1 with C++ programming language [4].

One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Because computer vision and machine learning often go hand-in-hand, OpenCV also contains a full, general-purpose Machine Learning Library (MLL).This sub-library is focused on statistical pattern recognition and clustering. The MLL is highly useful for the vision tasks that are at the core of OpenCV's mission, but it is general enough to be used for any machine learning problem [5].

IV. Procedure

In this section we will describe the development of our approach to completing the project. We will first describe the user interface that was developed. Next we will discuss our initial attempts to recognize sign language characters along with the problems we encountered, and our final algorithm.

A. *Resources*

Our platform consists of a common webcam that captures a 640x480 pixel image. Development was done on laptops with OpenCV-2.1 installed. All development was done using C++.

B. *Graphic User Interface*

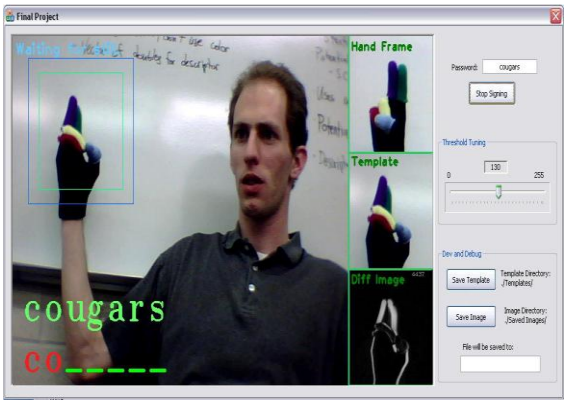
The presence of a graphical user interface (GUI) for other projects this semester has proved invaluable for testing and development. Easy methods for gathering test images to develop an algorithm and the need for various visual debug features necessitated the development of our own GUI for this project. The aim of this GUI was not to be a final product, but more of a debug and development environment to aid in the progression of our algorithm.

We started the GUI by reusing some of the code used in previous team projects. The portion of code that proved to be the most useful was the portion that controlled the display of the image data captured by the camera. Controls to enter text to prompt the user to sign and other controls to handle the collection of test data were then added.

Visual debug windows, as shown in **Figure 1**, are shown to the right of the captured image. To make this portion work was a challenge, for the model code provided from earlier project was custom fit for a normal sized image. These debug windows needed to be drawn onto the bigger display pixel by pixel. Further development would allow us to take advantage of more efficient OpenCV functionality.

One thing that we added to the GUI that was different from the previous projects is how the images are captured from the camera. The code that we were provided captured images continuously in a loop inside another thread as cameras provided them. We replaced this with a timer to capture an image about every 30 milliseconds to give us a more predictable frame rate.

Figure 1 – The final working version of the GUI



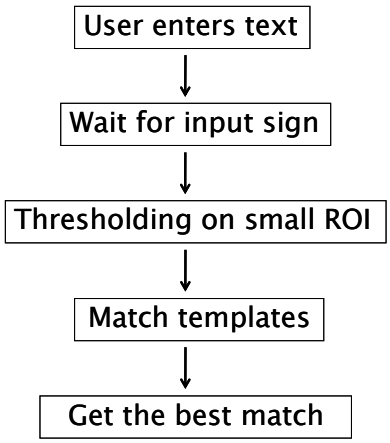


Figure 2 – Processing flow

C. *User Input*

Figure 2 describes the process the program goes through to get input from the user. After the user has entered text and indicated that it is time to sign, we detect motion on a small ROI where the hand is supposed to be making the sign. This ensures that we only start looking for a sign when the user is ready.

Then after a moment of stillness, we capture a frame and find the best match from all of the templates we took using the GUI. The result is printed to the screen and we start the process again by waiting for motion, repeating the process until the user is done entering text.

Figure 3 – An early template showing the number 7

Figure 3 shows and early example of a template that we used. We compare the templates against a slightly bigger region using the OpenCV matchTemplates function and return the best fit.

C. Preliminary Attempts

Our initial attempt to identify the proper hand sign was focused on template matching on a binary image. Templates were created from images of a bare hand that were then run through the OpenCV threshold function to produce a binary image such as that in **Figure 3**. This method proved to be extremely difficult to achieve a sufficient level of accuracy in identifying the proper signs. Many of the signs we are matching are too similar to be distinguished in a binary image and the difference in return values from the matchTemplates function were all extremely similar even if the proper template had the best value. We tried a variety of different thresholding levels and other refinements but found that we simply could not achieve the accuracy we needed with this method.

We also tried the openCV shapeMatching function on the binary images. Shape matching is based on contours and thus has an advantage over the template matching method in that it is more resistant to rotation of the templates. In template matching if the image was of a hand of the exact same shape as our template but was rotated, the template matching algorithm would not find a high correlation between the two images. Despite this advantage of shape matching we found that there was little improvement in using this method with the binary images.

Figure 4 – A latex glove with colored marker

Figure 5 – Glove created from several other gloves

We determined that in order to gain greater accuracy in matching our images against templates that we would have to use color images. This would provide us with more information to distinguish between signs. In order to make the different parts of the hand more distinguishable we choose to use a special glove that would have different colors for each finger. We hoped by having each finger more distinct that even signs with similar shapes would produce distinct images.

The first glove that we used was created by using colored markers on a white latex glove. This glove has the advantage of being able to put a different color for the front and back of each finger (provided you have enough markers). However, we discovered that it was difficult in the lighting conditions we had to make the colors bright enough to be distinguished as seen in **Figure 4**. We made some attempts to resolve this problem by using different techniques in processing the images, and it is possible that we could have resolved it by making greater efforts to control the lighting environment, but ultimately we decided to try a different glove.

Our next glove was created from several gloves of different colored cloth by cutting the fingers off some

gloves and putting them onto a black glove as seen in **Figure 5**. This glove proved to be more distinct with the solid colors as we had hoped, but lost the advantage of having different colors for the front and back of each finger. This could have been achieved by sewing various pieces of cloth together, but ultimately was not needed.

Template matching on the color images with this glove proved to be much more effective than matching on binary images. However, as we added more templates to match against there were still problems with multiple templates being similar. In an effort to overcome this we tried converting the image to the HSV color space and matching on the different channels of that image. The hope was that by accentuating the color properties of the image we would allow for greater distinctions between images. All efforts with this technique, however, failed to provide any improvement in accuracy. We also tried doing comparisons of the color histograms of the images but found that using the histogram removes all information about the shape of the sign and the difference in the amount of color visible between different signs was insufficient to distinguish them. Our final algorithm required slightly different techniques to get the accuracy we wanted as detailed in the next section.

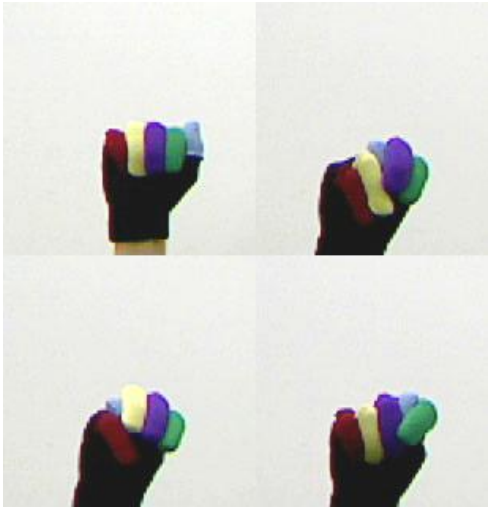


Figure 6 – Similar signs ‘a’, ‘n’, ‘m’, and ‘t’

E. Final Algorithm

Our final algorithm relied on template matching of colored images using our colored cloth glove as described above. In order to resolve our final problems of accuracy, however, we had to further constrain our project and decided to make the template signs more distinct from each other. For instance, in ASL the signs for the letters ‘a’, ‘n’, ‘m’, and ‘t’ are extremely similar, as shown in **Figure 6**, with the only difference being the placement of the thumb in between different fingers. Rather than rely on possible image techniques to distinguish between the small differences between these signs we decided to make the signs themselves more distinct by adding a distinct position in the frame or rotation of the sign to make it easy to match against. Thus the sign for ‘a’ is at the top of the frame, while the sign for ‘m’ is with the hand tilted to the left of the frame. By forcing the signs to be more distinct we were able to easily detect these similar signs using just the template matching technique without further refinement.

One final adjustment had to be made in order for our system to properly identify signs to match the user entered password. In American Sign Language there are some signs which are shared (or extremely similar) between numbers and letters. For instance, the sign for ‘6’ is essentially the same as the sign for ‘w’. This poses no problem in sign language because it is easy to use the context of a word to determine whether the sign is supposed to represent the number or the letter. You would not expect someone to sign that they are “4w” years old, so you know the sign must have been “46”.

In our system we took a similar context based approach to resolve this problem. Since our system is trying to decide if the user signed input matches a predefined password we can use that password as the context check for the input sign. Thus if the password contains a ‘6’ as the second character and the second sign the user gives is the sign that can be either ‘6’ or ‘w’ then it is assumed that the sign represents a ‘6’ and it is accepted. The user should be aware of the 5 existing signs for which there are two interpretations because since the system assumes the correct sign based off of the password there is some small loss in security when using these characters in the password.

V. RESULTS AND ANALYSIS

Using the final algorithm we were able to successfully indentify all signs in a number of test passwords. However, our system is highly constrained and is limited in its applicability. The template matching technique requires that the system have a set of templates that represent the same environment as the user input (e.g. same background, lighting conditions, etc.). Thus if a user took template images at home and then tried to use the program at work, it may not work properly if the environment was sufficiently different.

The user signs must also be sufficiently similar to the template images, especially with the modifications mentioned above concerning making similar signs more distinct by placing them differently in the frame. This however, could be seen as a security feature because someone unfamiliar with the proper signs to input would be unable to input a proper password. In fact, the user

could create their own set of signs not based on American Sign Language for the template images and the system would work the same. In this way, only they would know how to input the proper signs for their password, making the system more secure than conventional keyboard input.

Our present method of template matching on color images requires a large amount of processing time. Processing the binary images was much faster, but lacked accuracy. The additional information color gives to uniquely identify the signs is necessary for our accuracy requirements but requires extra processing time to process all three color channels. It may be possible through further tuning to process only a single color channel (either in RGB or HSV) and achieve the same level of accuracy with better speed. In our experimentation the saturation channel of an HSV image seemed most promising in this respect.

VI. Future Work

The system represented in this paper is primarily a proof of concept system. There is additional work that could be done to make the system more robust and accurate as well as more user friendly. One possible method for improving accuracy is the combination of various processing methods. For instance, it may be better to use template matching and histogram comparison and combining the results of these two methods to form a final answer. Care should be taken though as additional methods may add to the computation time necessary. However, it may be possible to reduce computation time by using the combined results of two much faster but less accurate methods rather than a single slower method. It may also be possible to use Haar training methods for this system, similar to those used for face detection. Given the number of templates to match on this technique seems unlikely to be effective, but perhaps deserves further investigation.

References

[1] Starner, T.; Weaver, J.; Pentland, A.; , "Visual Recognition of American Sign Language using Hidden Markov Models", Massachusetts Institute of Technology.

[2] Starner, T.; Weaver, J.; Pentland, A.; , "Real-time American sign language recognition using desk and wearable computer based video," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol.20, no.12, pp.1371-1375, Dec 1998

[3] Intel Corp.; , OpenCV Documentation Wiki, Accessed April 2011. http://opencv.willowgarage.com/wiki/.

[4] Wiki Entry; OpenCV, http://en.wikipedia.org/wiki/Opencv.

[5] Bradski, G.; Kaebler, A.; "Learning OpenCV – Computer Vision with the OpenCV Library", O'Reilly, 2008.