The Applications of Image Processing Techniques to Sign Language Recognition

Byron Hood

October 29, 2007

Abstract

In the 18th and 19th centuries, as the typewriter was gradually invented and perfected[6], the keyboard came into more and more use. However, the first keyboards were ineffective because of the way typewriters were then constructed: the levers would strike the page below the field of view, and page would scroll up later. This design was very prone to jamming, and therefore the Sholes & Glidden typewriters began to use the QWERTY keyboard (named for the first six letters along the top row under the numbers) layout optimized to slow down typing and keep common letters far enough apart to prevent jamming[6]. The same keyboard is still the most popular model)by a vast margin) today, and therefore people are greatly slowed in their typing even today. Although a popular way of speeding up is to use the speed-optimized Dvorak layout, this project explores a different idea: using image processing techniques to recognize and "understand" spelled-out sign language.

1 Introduction

1.1 Purpose

The purpose of this project is to provide an interface for people who speak sign language (whom I call "sign speakers") to input into a computer using their native form of communication. Such input should theoretically increase their input speed twofold, as explained in the Background section. In addition, such interpretation of sign language gestures and hand positions is a first step towards fulfilling programmers' dream of human-computer interaction nearly as old as the machines themselves, natural language processing, or the concept of a computer "understanding" naturally spoken (or signed) language.

1.2 Scope

This project will require some research into sign language hand positions and gestures, but more specific and deeper research is necessary into "image parsing" techniques (such as edge detection, line-finding, and so on). Additionally, the program will attempt to insert characters into the computer's input system, and this will require some heavy digging into either windowing system input code (so that the program can feed into X windowing system input) or operating system-level input (so that the program can emulate a keyboard). Data on hand positions will be stored in XML files read and parsed during the running time to ease the process of editing hand positions and also to make reading the said data simple when starting up; yet the project will also need some basic research into XML parsing schemes and methods of storing the XML data in memory.

The final application will recognize and interpret only alphanumeric characters for reasons of complexity. The further the program goes, the higher the complexity, which follows an exponential path. With the addition of positions, the distinctions between different positions (and therefore different letters) become smaller and less easily made. Therefore, the first step is to detect a reasonable number of different characters, and then to extend this basic program later on.

2 Background

In today's society, people with auditory and locutory disorders usually opt to commincate using sign language, a silent variation on the local language which uses body language: gestures, mouthing, and hand/finger positions, instead of spoken words. Through extensive practice and use (as normal people might gain extensive practice speaking their native language), many sign speakers are capable of "speaking" as fast as non-impaired people speak orally, about 200 words per minute in normal conversation[2].

If the average word is taken to be six letters long and one accounts for a slight speedup due to the short amount of time required to communicate a single letter, spelling a word out letter by letter will likely reduce speed by a factor of four for both sign and oral speakers. Nonetheless, this is a hefty 50 words per minute, and compares quite favorably to average typing speeds. According to Karat, et. al. the average computer user can type 33 words per minute while copying text, and this drops to a mere 19 when composing[3]. Therefore, an average computer user will spend from two to three seconds typing any given word, whilst a sign speaker (could he or she sign into a computer), would spend half of that time. Finally, the "QWERTY" keyboard was designed expressly for the purpose of *slowing* typists down (this is a throwback to the days of typewriters, to help prevent jams). Therefore, a person signing has a double advantage over a person typing: first of all, they are not inhibited by the popular keyboard, and secondly, they sign faster than the average person types.

While extensive and highly specific research has been done in the field of computer vision (as shown by the sheer number of books available on the subject), little has been devoted to the recognition of sign language, and none has considered using a webcam-computer setup (modern research explores using a specialized glove to transmit data back to the computer). This is for a combination of reasons: first of all, processor power was formerly far too expensive and not powerful to process a multitude of images (with a high enough resolution to distinguish sophisticated shapes such as the human hand) in anything close to real time. Additionally, the keyboard has been—and remains—an effective, flexible, cheap, and easily extensible tool for computer input. Finally there is as of yet little demand for such a novel mechanism of input.

Nonetheless, I have reviewed some background material, especially in image processing, much of which explained techniques such as Hough's transform for finding shapes[4], Robert's Cross, and Sobel's Operator (both edge detection methods). As very little, if any, work has been done in this field, I am pretty much a pioneer—also an inventor—and

I must decide what path to follow on my own with little outside guidance from previous products. Despite this, I will still search for similar projects, and if these exist, I will attempt to model my path based on theirs, with a new twist to make the project interesting.

3 Procedures

The project is most easily described in steps; the breakdown of the program's core elements listed in the order in which they will be run for each frame processed. Each runs in parallel to the others and does its job asynchronously. Therefore, each program will simply grab the latest output from the previous program and run on based on that.

1. **Capture the frame:**
   The program will use the Video for Linux 1 ("V4L") application programming interface ("API") to connect and capture frames from a webcam attached to the computer. After capturing each frame, it is saved to one of a set of files on the hard disk and the program loops (until it receives the signal to exit).

2. **Edge-detection:**
   Running alongside the capture program, an edge-detection program loads the data saved by the capture program asynchronously and runs the data through the Robert's Cross algorithm to generate a second image which only contains the edges of the first image. At this point, the first image is discarded; the edge-detection program passes the data on to a line-finding program and then returns to the top of its main loop.

3. **Line-finding:**
   The line-finding program takes data from the edge-detection program in the form of an image which contains only the edges in the originally-captured frame. Then, it uses Hough transforms[4] to identify and parameterize any and all lines in the image. The resultant data is sent off to the next program and the line-finder loops.

4. **Line Interpretation:**
   Using the lines found in the last program, this is in reality an AI which uses a heuristic to attempt to piece together a digital representation of a hand. Once this is put together, it determines in what shape the hand is (e.g. is the index finger straight up or curled all the way in?). After deciding upon a specific position for each finger, this program loads data from a set of XML files which parameterize positions of the hand for each letter of the sign alphabet (plus numbers in sign). It then proceeds to match the hand detected to the preset hand position for each letter. If a match is found, then the result is inserted into standard input (or if my time is limited, it will just print the letter out).

The order in which this will be programmed is a question of making each piece as testable as possible as quickly as possible, so that large amounts of basic-level debugging are not necessary after the very beginning. The first (now completed) step will be edge detection, followed by shape (and finger) detection from the data produced by the edge detection program. After this, I will write the program which will use the data from the XML files to match against known data for certain alphanumeric characters.

3.1 Testing

To test the program to make sure it is properly working, I will apply separate, specialized tests to each portion, reasoning that if each individual parts works, then any problems in the whole must be caused by joining the pieces. For example, I will make sure that the video capture program properly saves an image in the proper format in the proper place before using the data it saves for any purpose. The alternative, writing a program to check on the performance of my other programs, both in terms of what they are supposed to do and also their timeframe, would be impractical as this would require image analysis (and what I am currently writing heavily involves image analysis).

3.2 Software

The program will be written exclusively for a Linux platform, due to the lack of a good (free) C compiler for Windows. My decision to use C as a programming language was somewhat arbitrary, but also grounded in statistics: if we start with the entire universe of computer languages, we can immediately eliminate any interpreted lanugages (such as Python) as these are too slow (a Python edge-detector ended up taking 20 seconds compared to 100 milliseconds for a comparable C program). Further, among compiled languages, Java is unfeasible because the Java VM requires too much processor and memory, Fortran has no decent free compiler, and older lanugages such as Ada are not well-enough documented. This leaves me with C and C++, and among the two I chose C because of a small speedup which will hopefully manifest itself over the millions and millions of calculations this application will carry out.

3.3 Algorithms/Programs

To detect edges in frames captured from the camera, I will be using a variety of edge-detection techniques, including Robert's Cross and Sobel's operator. These are primarily edge-detection algorithms and will allow me to print out balck-and-white images displaying all of the edges in an image, now in a format that is much easier to work with. For detecting lines and shapes in the black-and-white image remaining after edge detection, I will use Hough's transform[4].

4 Schedule

My schedule is one of landmarks to be achieved rather than a very specific, fixed one which allocates time for each specific task. Such a schedule, I believe, should ease pressure so that if I fall behind in one place but make extra headway in another, I can still congratulate myself in good conscience and consider myself "on schedule." A rough timelien fo what I want to have accomplished after each quarter:

4.1 First quarter

- Write a functional camera communication program (though it need not be perfect) which grabs frames and saves them to disk.

- Implement an edge detector using the Robert's Cross algorithm (this should be very well-tested).
- Begin a cropping function for edge-detection and start a line-finding program.

4.2 Second quarter

- Finish the cropping functionality of the edge detector.
- Continue work and testing on the framegrabbing program, complete it if possible.
- Continue work on and (possibly) finish the line-finding program.
- Begin the line-interpretation AI.

4.3 Third quarter

- Finalize the framegrabbing program in both coding and testing.
- Finish and thoroughly test the line-finding program.
- Finish and test the the line-finding AI.

4.4 Fourth quarter

Finish and test any unfinished programs, and then unify the entire application.

5 Expected Results

Should this endeavour succeed, the results will be beneficial to all people with either hearing or speaking disabilities, or those that interact with them as they will have the option to "talk" naturally with their computer (albeit spelling things out). While this program may well become a first in its field, it can certainly be of use to further researchers and/or programmers who may attempt to extend it or write more complete interfaces which encompass more than simple hand positions for alphanumeric characters (and perhaps include special symbols like '$', '%', or '"'. The results will be best shown in practice, by demonstrating the program or having fluent sign speakers test it out. The returns from such tests could be very easily graphed; for example the accuracy versus signing speed, or accuracy over time, and so on.

References

[1] Brown, C. M. (1988). Human-computer interface design guidelines. Norwood, NJ: Ablex Publishing

[2] Omoigui, N., He, L., Gupta A., Grudin, J. and Sanocki, E. (1999). "Time-compression: Systems concerns, usage, and benefits." *Chicago 1999 Conference Proceedings*, 136-143.

[3] Karat, C. M., et. al. “Patterns of entry and correction in large vocabulary continuous speech recognition systems.” *Chicago 1999 Conference Proceedings*, 568-575.

[4] Foregger, Thomas. “Hough Transform.” 06 Aug 2006. PlanetMath. 28 Sept 2007. <http://planetmath.org/encyclopedia/HoughTransform.html>

[5] <http://paginas.terra.com.br/informatica/gleicon/video4linux/videodog.html>

[6] “Typewriter.” *Wikipedia*, <http://en.wikipedia.org/wiki/Typewriter>