Using Gesture Recognition to Navigate Google Chrome

Yuki Adams
Department of Computer Science
Earlham College, Richmond, IN 47374
ybadams15@earlham.edu

ABSTRACT

This hand gesture recognition system provides a convenient tool for Google Chrome users to navigate through the browser on a personal computer. It allows a user to scroll, zoom in and out, and move forward and backward a page by using a handful of static gestures. The system utilizes the built-in camera, and the live video feed is preprocessed. The simplified frames are passed into a k-nearest-neighbor classifier, which has been trained with 300 images (50 images for each static gesture). Each frame is given a predicted label based on the target labels from the training set and the value of k. The predicted label is passed into the Google API, where each label is mapped to its associated navigational function.

KEYWORDS

gesture recognition, computer vision, supervised machine learning

.

1 INTRODUCTION

When humans communicate with each other, they often use hand gestures and body language to help convey their verbal messages [9]. When humans interact with computers, some methods of human input include a mouse or touchpad, keyboard, and speech. In this paper, I implement another form of human-computer interaction (HCI), which utilizes a standard laptop camera and software. The camera provides real-time video feed such that an individual can produce various hand movements that correspond with different functions, all without touching the computer. The aim of this project is to use these hand gestures to navigate through Google Chrome, a widely used web browser. Such navigation includes scrolling, zooming, and moving forward or back a page in a particular window.

Gesture recognition has real world applications. One of the important roles of technology is to make life more convenient for humans. Gesture recognition systems have this characteristic, due to the intuitiveness of many hand gestures. Currently, gesture recognition is being employed in virtual/augmented reality (VR/AR) systems, such as Nintendo Wii's Wiimote [19] controller, and Microsoft's Xbox Kinect [12]. However, the hardware devices used in these products are expensive and have limited applications – mostly within the entertainment sector.

Due to financial constraints, many people can not access these technologies. On the other hand, many people do have laptops and computers, and use browsers like Google Chrome in their everyday lives. Gesture recognition systems that use a basic camera can exploit this accessibility and provide a cheaper method of gesture-based HCI.

This paper outlines the implementation of a gesture recognition system to navigate Google Chrome. Python was used to implement this software, along with OpenCV, an open-source library that focuses on computer vision tasks [3]. Initially, the system takes the camera's video feed to detect and isolate the user's hand in a region of interest (ROI) within the camera's video dimensions. After this stage, a machine learning model that uses K-Nearest-Neighbors classifier will predict a given gesture, based on the set of training gestures. Upon accurate recognition of a gesture, the gesture's label (e.g. "zoom out") will be read into Google Chrome's API, to simulate the clicking of the back button to navigate back a page.

2 BACKGROUND

Gesture recognition is a relatively young field of research, but the importance of using gestures to interact with a computer is great. Gestures remove the need to touch your device and can be read from a distance. These features are very useful if a user's hands are dirty. Gestures are already commonly used among people, so there are gestures that are intuitive. This means that it is very simple to learn and easy to use [16].

Gestures can be broken up into two categories. There are static gestures, or postures, that are still images of the hand in a certain configuration (e.g. the index and middle fingers extended, other fingers balled up into a fist). Static gestures can be recognized in numerous ways. Conversely, there are dynamic gestures, which are gestures that involve physical movement, and can be identified with machine learning and deep learning. These gestures are a sequence of images of a hand performing a gesture. In this project, I use static gestures as input.

CS488 Senior Capstone, Dec 2018, Earlham College

ACM ISBN . . . $15.00

2.1 Hardware Devices

Although gesture recognition is not something prevalent in the homes of everyone, much progress has been made in the way of utilizing gestures in technology. The Xbox Kinect is an accessory to the Xbox One gaming console. It is a sensor that sits above or below the TV, and can recognize individuals, as well as detect the user's hands. The primary use in this context is to play games on the Xbox console [12].

Another commercial hardware system is the Leap Motion Controller. Leap Motion has built a device that contains all of the sensors and capabilities for gesture recognition, but it was made for developers so they could create games or other types of applications [1, 2, 5, 15]. Unfortunately, a general consumer may not know how to program with the Leap Motion Controller.

Finally, Google has been developing Project Soli, which is a microchip that contains the hardware for fine, precise gestures. Google utilizes radar technology to accurately sense any gesture within its range. However, this project is still in development and has not yet been commercialized. Contrary to Google's robust hardware, gesture recognition on a laptop doesn't need to be so precise, as a majority of the gestures being used in this project are forms of waves rather than small finger gestures, which is what Project Soli has been built to recognize [8].

2.2 Software

The software development in gesture recognition includes OpenCV, the primary library that is used in gesture recognition [3]. OpenCV is used for a variety of computer vision tasks, including image processing and facial recognition. Another library that provides computer vision tools is Python's Scikit-learn library [17]. Similar to OpenCV, Scikit-learn provides methods for manipulating images and models for facial and gesture recognition.

Another development is the use of supervised machine learning methods to recognize gestures. One of the more frequently used machine learning techniques to recognize dynamic gestures is the Hidden Markov Model (HMM) [13]. The HMM is a flexible tool used for modelling time series data, and is prevalent in many applications, such as computational molecular biology, data compression, artificial intelligence, and pattern recognition, to name a few. The problem of dynamic gesture detection and classification contains time series data, and these gestures can be represented as probability distributions over a sequence of images. This mathematical function provides probabilities of the occurrence of different possible outcomes. In the case of dynamic gesture recognition, these different outcomes are the possible directions a specific hand posture could move over a sequence of frames in a video. The technique gets its name from two specicfic characteristics. The model is "hidden" because it assumes that the observation at time t is generated by a process, and the state of this process is hidden from the observer. It is denoted as a "Markov" model because it satisfies the Markov property. This means that at any given frame during dynamic gesture recognition, the current state of the model encapsulates all of the information it needs to know about the states before it, allowing the model to predict future processes [6].

Contrary to dynamic gesture recognition, a classifier can be used to recognize static gestures. The classifier being used in this gesture recognition system is the k-nearest-neighbors (KNN) classifier [17]. KNN classification uses a simple algorithm that stores a training set of data and classifies new data – in this case a frame from the video feed – based on its similarity to the k nearest data points. The training set is divided up into several target labels (or classes), and the label given to the new data point is based on what the labels are in the k nearest data points. In other words, KNN classification works based on the new data point's feature similarity with the training set. The predicted label is given based on the majority of the k nearest data points.

KNN has been used in few image classification applications. A group of researchers from India have used KNN classification to classify images of flowers. The researchers created a set of images of flowers in varying light conditions with a cluttered background and various poses, and used texture features to classify flowers [7]. KNN has also been utilized in optical character recognition (OCR), which is classification of printed or handwritten characters [14]. The researcher, Moghnieh, lays out a method to recognize 500 scanned images with noisy backgrounds with a 98 percent accuracy. He explains his process of OCR using a histogram of oriented gradients (HOG) as the feature set of the images, and performs KNN with Scikit-learn's KNeighborsClassifier [17].

The reasoning for why I chose to use a KNN classifier is because this gesture recognition system has a limited "vocabulary." In other words, there is a small set of gestures, each varying from each other to a great degree. Since the gestures will be drastically different from one another, classification of a gesture should be accurate without a more complex method.

3 METHODOLOGY

In this project, there are five modules that the data flows through. The live video feed that starts the pipeline originates from the built-in camera. The framework is shown in Figure 1.

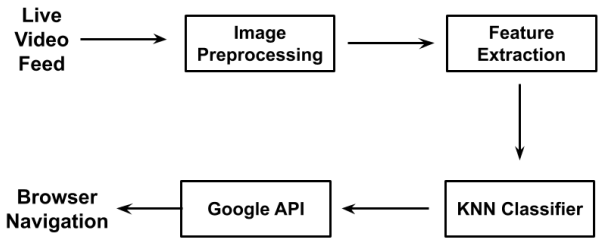


Figure 1: Design Framework

3.1 Image Preprocessing

Prior to image preprocessing, a region of interest (ROI) is defined for performing the gesture recognition. Since most people are right handed, the ROI is offset to the right side of the frame. The size of the ROI is 300x300 pixels, which allows a user to perform gestures at a comfortable position and distance from the camera. Additionally, this reduces the size of the array of pixels greatly, from an entire frame to just a subset of the frame, which also reduces the amount of computation needed. The gesture recognition is performed only within this ROI. The ROI is shown in Figure 2 below.

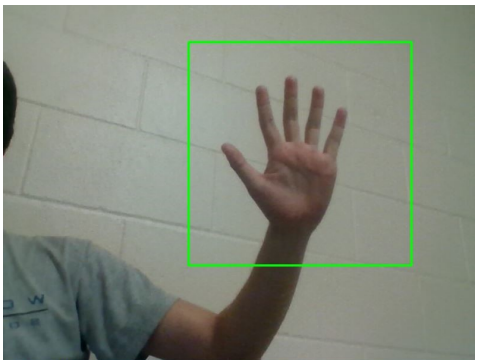
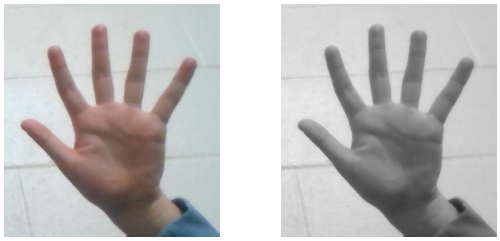


Figure 2: Green box represents ROI.

3.1.1 Running Average. As the live video feed is being read from the built-in camera, each frame undergoes preprocessing. The first step is to compute the running average of the ROI. In other words, the background becomes separated from the foreground (hand gesture). This makes it easy to identify what is the hand and what is not. A global background variable is declared, followed by a running average function that takes the first frame of the video feed, before a hand is moved into the ROI, and defines it as the background. Then, using an OpenCV method called *absdiff*, an absolute difference is calculated between the background and current frame containing the user's hand, thus separating the hand from the background.

3.1.2 Color Conversion. In the next step, every frame is reduced to the dimensions of the ROI using image slicing, then converted to grayscale using OpenCV's COLOR_BGR2GRAY method. This grayscale image reduces the original frame from an array of pixels with red, green and blue (RGB) channels for each pixel to an array of pixels with a single value for each pixel (this value represents the brightness of the pixel, i.e. how white a pixel is. The conversion from color to grayscale is shown in Figure 4 below.



(a) Static hand gesture. **(b) Grayscale.**

Figure 3: Original Frame to grayscale.

3.1.3 Binary Mask and Blurring. The running average function helps separate the hand from the background, but further processing is needed to get a clear image of the hand. A binary mask, or conversion of an image into black (0) or white (255) pixel values. OpenCV provides two methods, called THRESH_BINARY and THRESH_OTSU that take a grayscale image and convert it into a binary mask via thresholding. In thresholding, any pixel with a value lower than a specified threshold is reduced to black, and any pixel with a value greater than the threshold is increased to white. THRESH_OTSU is a smart method of thresholding that determines what the threshold should be. In a bimodal image, that is an image with a grayscale histogram that has two peaks, THRESH_OTSU will place the threshold value between the two peaks, providing an optimal threshold value. With this method, the binary mask may still contain some noise that must be smoothed out. To do so, a Gaussian blur is applied using OpenCV's GaussianBlur function. The binary mask of a gesture is shown before and after Gaussian blurring in Figure 4b below.

(a) Binary mask. **(b) Blurred mask.**

Figure 4: Original Frame to grayscale.

3.2 Feature Extraction

Now that there is a blurred binary mask of the hand, the features must be taken from the preprocessed ROI. By using blob detection, we take information from white blobs in a binary image and convert it into a few representative features. Blob detection is the process of confining blobs (or connected white pixels) in a binary image inside a bounding box or circle [4, 11]. OpenCV contains a SimpleBlobDetector_create function that can filter a binary image based on several parameters, such as blob color, area, inertia, convexity, and circularity. The blob detector will try to bound the blobs

based on those parameters. Multiple blobs may be detected in a binary image, but only the largest blob (hand) will be used for calculations. One important feature is the area of the convex hull, which is the minimum convex polygon that contains the BLOB. Another feature used is the centroid, or center of mass. The last feature extracted is the circularity of the BLOB, which is the ratio of the perimeter of the BLOB to the perimeter of a circle with the same area. Every frame will be reduced to these features. In Figure 5 below, a bounding circle and the convex hull are drawn.

Figure 5: Binary image with detected blob. Bounding circle and convex hull are drawn.

OpenCV provides methods that give the coordinates of the centroid as well as diameter of the bounding circle. These are used to calculate the three features being stored for classification of each frame.

3.3 KNN Classifier to Google API

In order for classification to work, I had to create a labelled dataset for each gesture. To do this, I wrote a script that would record live video for two seconds, preprocess each frame, and store the binary images in separate directories. To avoid having duplicate images of gestures, I moved my hand around inside the ROI to simulate variations of a particular gesture. I created eight directories, one for each gesture, then wrote each image to the appropriate directory. This created 60 images per gesture, for a total of 480 images in the dataset.

To extract the features from each image, I used two methods. In the first method, I simply reduced the image to one-third of the original size using image slicing. The dimensions of each image went from 300x300 pixels to 100x100 pixels as a result. OpenCV represents grayscale and binary images as 2-dimensional Numpy arrays, and Numpy provides a method called *flatten*, which takes a 2-dimensional Numpy array and converts it to a 1-dimensional Numpy array. After flattening each image, they were all labelled based on the directory they were stored in. Finally, the flattened arrays were passed into a Pandas dataframe for classification. In the second method, circularity, convex hull, and centroid for each image were calculated using OpenCV methods and passed into a dataframe. The resulting dataframe contains only three features for each image, rather than all the pixel values for each image as described in the first method.

Scikit-learn provides a KNeighborsClassifier function and train_test_split function that passes a labelled dataset into the classifier, and splits the dataset based on specified parameters. This results in a training and testing set that can be used to test the accuracy of the classifier. By using various splits (50/50, 30/70, etc.), the classifier can be trained with various training sets. Additionally, the value of k can be tuned to yield the greatest prediction accuracy.

After the testing phase is complete, the final step will be to implement the classification within the code that performs the gesture recognition in real time. The training set will become the entire dataset that I created, and each frame will be passed to the trained classifier as a testing set. The classifier will then output a predicted gesture label any time a user's hand is within the ROI.

Finally, this predicted label is passed to the Google API. The label is passed to Google Chrome's command API, which allows users to develop their own keyboard shortcuts. So, if a predicted label is "point left," the text passed to the API will be "Alt + Left-arrow." The window will act as if the user pressed Alt and the left arrow key (move back a page). The functions include scrolling up and down, moving forward or backwards a page, and zooming in and out. With these basic navigational functions, a user will be able to navigate a webpage on Google Chrome efficiently.

4 RESULTS

Currently, the testing phase is still in progress. However, the first feature extraction method mentioned above has been tested several times. By splitting the dataset in half, one half can be assigned as the training set and the other the testing set. By using Scikit-learn's accuracy_score metric, the accuracy of prediction can be determined. With the dataset that represents every pixel as a feature, the KNN classifier yielded accuracy in the range of 99.69 percent to 100 percent. The time it takes the dataframe to fit the KNN classifier is approximately 14 seconds, and the prediction stage takes less than one second. The implications of the results will be discussed in the following section.

As I have yet to test the second method that uses a dataset of three features per image, no results have been determined. However, the expectation is that the accuracy will not be as high as the results from the first feature extraction method due to the loss of information stored from each image. However, the time it takes to fit the KNN classifier should be reduced, as the number of features per image is significantly lower than in the first method. After testing the second method of feature extraction, the method that yields the fastest results will be used to recognize gestures in real time.

5 DISCUSSION

The first method for feature extraction yields near perfect accuracy when testing. Most likely, this is due to the lack of

noise in the image dataset. When creating the image dataset, the background was dark, in contrast to my hand. Additionally, a flashlight was shone onto the hand to increase contrast between the foreground and background. This allowed for very little noise, if at all, to be detected in the binary mask preprocessing stage. Although each image for each gesture label is slightly different from one another, reducing the size of the image by a factor of three may increase homogeneity of the images in each class.

KNN classification is a lazy machine learning algorithm, meaning that it stores all of the data from a dataset, but doesn't perform any computation until the prediction stage. As such, it was expected that the prediction stage would be relatively low. In order for prediction to work in real time, the preprocessing stage and the prediction stage would need to take less than or equal to one-thirtieth of a second, as the live video feed on the local machine plays at 30 frames a second. Unfortunately, it is expected that the preprocessing of an image may take longer than a small fraction of a second, even if the prediction relatively quick. So, in order to create a real-time gesture recognition system, the preprocessing stage must be reduced.

The mapping of each frame's predicted labels did not perform as expected. There were many difficulties with Google's API. One such problem involved the use of the current window. The webpage couldn't be accessed without the use of a Google Chrome extension. Another issue that came up was that certain Google Chrome keyboard shortcuts couldn't be overwritten. For example, a predefined keyboard shortcut like copy (Windows: Ctrl + C, Mac: Cmd + C) could not be mapped to a different keyboard shortcut.

6 RELATED WORK

There have not been many gesture recognition systems that have been used to control a browser. One project uses Anaconda (Spyder IDE) and OpenCV to develop a system that uses gestures to automatically open up frequently used webpages on Google Chrome [10]. Unfortunately, this project is very limited in its ability to control the browser. This browser controller simply allows a few gestures to be mapped onto different webpages, and it will open up the webpage in a new tab. This feature, although time-saving, doesn't do much in the way of interacting with the browser. The proposed project will utilize OpenCV in Python for much of its framework also, but the browser control will be more interactive. Ideally, the proposed project will use Google Chrome's command API in order to change the browser navigational features, like scrolling, going forward/back a page, etc. to control the browser.

In another project, the developer used JavaScript and jQuery to create an extension that allows users to scroll through a webpage [18]. My proposed project will create a library of gestures that can be recognized, and ideally the program would be able to use these gestures for multiple functions within the browser, like scrolling. Although this project is similar to the project being proposed in this paper, there is only one browser function being utilized with gestures, whereas I hope to implement more features than just scrolling. Additionally, this project used preexisting code found on Github. Therefore, this project was more related to the creation of the Chrome extension. The lack of research in the field of gesture recognition with respect to browser navigation implies that this is a relatively new idea, and there exists room for development to provide users with a more multi-modal, interactive navigation for browsers.

7 FUTURE WORK

Gesture recognition can be used in just about any application, just as how a mouse or keyboard are used. However, they are very efficient, similar to how keyboard shortcuts or macros can perform functions that would be more time-consuming than using the mouse to navigate to perform a particular function.

One possibility for future work is to implement a Google Chrome extension that has all of the static gesture capabilities (as planned in this project), as well as a variety of dynamic gestures. In order to use dynamic gestures, the recognition system must be able to see the transition of a gesture over time, as explained in the Background section.

One of the possible future uses for this gesture recognition system is to open or run applications with it. For example, a user can perform a specific dynamic gesture to unlock a computer or other device. This is similar to facial recognition, but it doesn't have the same risks associated with it (e.g. you can't use a picture of a user performing a gesture). This is a more secure method for signing onto a device.

Another potential direction this project could move toward is use within an application. In other words, gestures can be mapped to an application's controls and used instead of the actual touchscreen, mouse, or keyboard. Gestures can be implemented within an operating system, like MacOS, and the gestures can replace keyboard shortcuts for navigation between browsers and applications.

Gestures are not limited to computer applications and gaming consoles. Gestures have already been utilized in mobile devices and embedded systems. For example, certain smartphones will turn the screen on when raised in a certain way. The screen will turn off when the phone is raised to the ear. Apple's Airpods can recognize a variety of gestures, like taking the earbuds out of the ear or putting them in. Music can be started or stopped based off these gestures. A smartwatch can turn its screen on when you flip your palm downwards. These are all examples of gestures being used in everyday devices. However, it can be taken further. Hand waves can be used to perform calling functions on smart devices. Even embedded devices can benefit from using gestures.

Outside of these applications, gestures can be used to learn or teach sign language. As more gestures are created and added to the library of gestures, these can be used by teachers to help students accurately perform sign language

gestures. Students can perform these gestures in front of a camera or other sensor to test competency and form of their sign language abilities. Gesture recognition can be used to facilitate the spread of sign language to individuals, and it could even be used in place of a computer keyboard (assuming a user's vocabulary is large enough).

REFERENCES

[1] Nicola Bizzotto, Alessandro Costanzo, Leonardo Bizzotto, Dario Regis, Andrea Sandri, and Bruno Magnan. 2014. Leap motion gesture control with OsiriX in the operating room to control imaging. *Surgical Innovation* (2014), 655–656. https://doi.org/10.1177/1553350614528384

[2] Alex Colgan. 2014. How Does the Leap Motion Controller Work? http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

[3] Intel Corporation. 2000. Open Source Computer Vision Library (OpenCV). https://opencv.org/about.html

[4] A. J. Danker and A. Rosenfeld. [n. d.]. Blob Detection by Relaxation. ([n. d.]).

[5] Youchen Du, Shenglan Liu, Lin Feng, Menghui Chen, and Jie Wu. 2017. Hand Gesture Recognition with Leap Motion. (2017).

[6] Zoubin Ghahramani. 2001. An Introduction to Hidden Markov Models and Bayesian Networks. *Hidden Markov Models* (2001).

[7] D. S. Guru, Y. H. Sharath, and S. Manjunath. 2010. Texture Features and KNN in Classification of Flower Images. *Recent Trends in Image Processing and Pattern Recognition* (2010).

[8] Lien J., Gillian N., Karagozler M. E., Amihood P., Schwesig C., Olson E., Raja E., and Poupyrev I. 2016. Soli: Ubiquitous Gesture Sensing with Millimeter Wave Radar. *The 19th Korea-Japan Joint Workshop on Frontiers of Computer Vision* (2016). https://doi.org/10.1109/fcv.2013.6485473

[9] Adam Kendon. 2004. *Gesture: Visible Action as Utterance.* Cambridge: Cambridge University Press, Cambridge University.

[10] Shivam Malani. 2018. Hand-Web Browser - Controlling Web Browser via different hand gestures. https://www.geeksforgeeks.org/project-idea-hand-web-browser/

[11] Satya Mallick. 2015. Blob Detection Using OpenCV (Python, C++). https://www.learnopencv.com/blob-detection-using-opencv-python-c/

[12] Microsoft. 2010. Kinect sensor for Xbox 360 components. https://support.xbox.com/en-US/xbox-360/accessories/kinect-sensor-components

[13] Sushmita Mitra and Tinku Acharya. 2007. Gesture Recognition: A Survey. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICSâĂŤPART C: APPLICATIONS AND REVIEWS*.

[14] Hussein Moghnieh. 2018. Scanned Digits Recognition using k-Nearest Neighbor (k-NN). https://towardsdatascience.com/scanned-digits-recognition-using-k-nearest-neighbor-k-nn-d1a1528f0dea

[15] Leap Motion. 2010. Leap Motion. https://www.leapmotion.com

[16] Wachs J. P., KÃűlsch M., Stern H., and Edan Y. 2011. Vision-based hand-gesture applications. *Commun. ACM* (2011). https://doi.org/10.1145/1897816.1897838

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[18] Dave Ross. 2014. Chrome Gesture Control. https://github.com/daveross/chrome-gesture-control

[19] SchlÃűmer T., Poppinga B., Henze N., and Boll S. 2008. Gesture recognition with a Wii controller. *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction.* https://doi.org/10.1145/1347390.1347395