Real-Time Gaze Tracking Using Webcam

Phyo Thiha

Abstract

Eye gaze indicates a person's attention over time, which is a powerful cue for determining what she finds interesting. Therefore, eyes have not only indispensable meaning for human communication, but also great potential to build human computer interaction in a more natural and direct mode. However, most available gaze trackers are either driven by specifically designed operating software or high end hardware[1,2,3,4]. High costs have always been the barrier prohibiting gaze tracking technology. Moreover, the majority gaze trackers have adopted corneal reflection tracking technique which actively illuminates the eye region by infrared (IR) light. Due to long exposure or close proximity to the IR source, eye hazards may arise. To overcome these problems, we developed a low-cost, webcam-based gaze tracking system, which tracks the gaze in nine equally-sized areas (see figure 1) of the monitor with the accuracy of up to 86%.

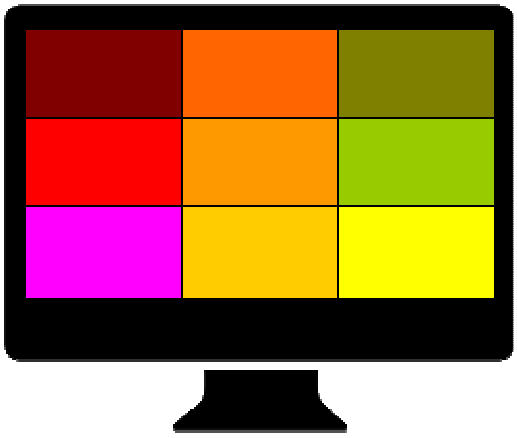


Figure 1. Our gaze tracker tracks the eye in nine equally-sized areas on a computer monitor

1. Introduction

Gaze (Eye) tracking is a technique whereby an individual's eye movements are measured so that the researcher knows both where a person is looking at any given time and the sequence in which their eyes are shifting from one location to another. Tracking people's eye movements can help Human Computer Interaction (HCI) researchers understand visual and display-based information processing and the factors that may impact upon the usability of system interfaces. In this way, eye-movement recordings can provide an objective source of interface-evaluation data that can inform the design of improved interfaces. Eye movements can also be captured and used as control signals to enable people to interact with interfaces directly without the need for mouse or keyboard input, which can be a major advantage for certain populations of users such as disabled individuals.

Due to its wide range of applicability, quite a few commercial and open-source gaze tracking systems are available. The ones that require special hardware--which is mounted on the head or

attached to body parts near the face of the user--are sold commercially and could cost as much as tens of thousands of dollars[4]. Most freely available eye trackers rely on the webcam, but some of them do not work with reliable accuracy[5,6], and some depend on third-party code libraries[7,8], which are outdated or are extremely difficult to install. These observations motivated us to develop our gaze tracking system such that it is affordable and deployable with very little dependency on third-party software. Our system only requires a run-of-the-mill webcam, and runs on any operating system with just two open-source Python libraries needs to be installed[9,10].

In the following sections, we present the background on different gaze tracking techniques, presentation of our gaze-tracking model, approaches we experimented and their results, and discussion about the strengths and weaknesses of our system. Next, we provide conclusion and suggestions for future work.

2. Related Works

A variety of eye-gaze (eye-movement) tracking techniques have been reported in the literature [11, 12]. We can broadly classify non-intrusive, eye tracking techniques into two major categories[13]. The first category is based on directing a beam of light (typically infrared) into the eye and then measuring the light reflected from the eye. The second category is based on measuring the gradient of the skin around the eye. There are about four eye tracking techniques which uses the reflecting light from the eye: Limbus tracking (the boundary between the white scalera and the darker-colored iris)[14], pupil tracking (the boundary between the pupil and the iris)[15,16], Purkinje tracking (reflections from the boundaries of the lens and cornea)[17] and Corneal reflections tracking[18]. Our approach closely resembles that of Limbus since we extract the entire area enclosed by the iris for tracking.

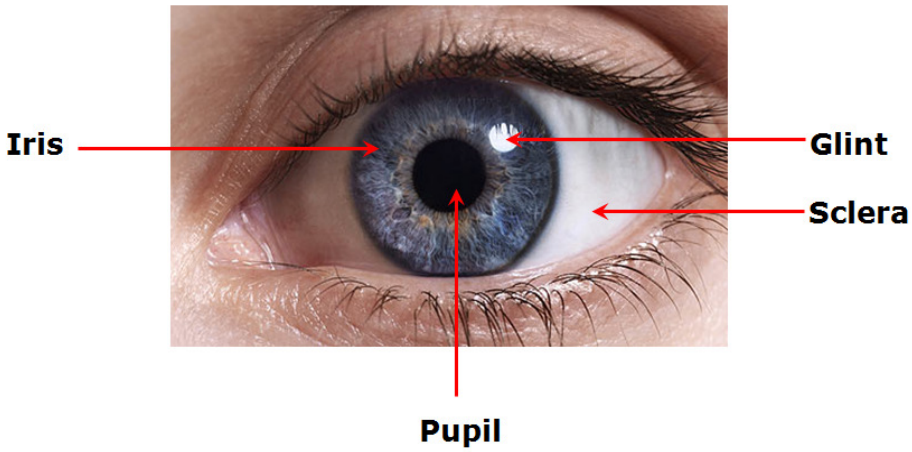


Figure 2. Terminologies of a human eye. The lens sits inside and behind pupil and the outermost layer in front of the iris and the pupil is called "Cornea" whereas that in front of the white area is called "Sclera". Due to corneal reflection, there is a glint from the cornea of the eye as long as sufficient light is present in the environment.

There still have not been many robust gaze tracking systems designed to work with inexpensive webcams [19]. The video based systems are found to require high resolution images of the eye to accurately determine the point of gaze (POG). Ohno et al. developed a single camera system which achieves accuracy under 1 degree of visual angle [20]. However, their approach requires the camera to be mounted at 60cm distance from the eye. In contrast, our system relies on a fairly grainy resolution from the webcam, which is attached to the computer monitor. This allows the user to not be physically constrained, and the distance between her eye and the camera to be tolerant up to a couple of inches.

One of the early non-intrusive, camera-based gaze monitoring system utilizes neural networks to compute the eye gaze[21]. In order to locate the eyes, the system searched for normalized skin color of the user's face in the image. Once the face is detected, the algorithm focuses on searching for pupils—assuming the geometric constraints of a human face—by iterative thresholding, which also normalizes the various lighting conditions in the environment. Once the pupils of both eyes are detected, regions with a size of 20x10 pixels—centered on each of these pupil locations—are used as input to train the three-layered neural network with backpropagation. This neural network based approach yield the accuracy of between 1.4 to 2.0 degrees. Inspired by their success, we also deployed artificial intelligence and machine learning techniques to determine the gaze direction.

3. Hardware and Software Setup

We used Logitech HD Pro Webcam C910. The camera is capable to take full high definition video (HD 720pm) with screen resolution of 1280x720. As of May 2012, it is priced at $71 in multiple online retailers. Our system requires the webcam to be attached to the center of the monitor and the user be sitting in front of it (see figure 3). As long as the user's face and eyes are visible in the camera's output—which is indicated by a red box drawn around the left eye of the user (see figure 4), the system is ready for training and use.

Figure 3. The webcam is set up approximately at the center of the monitor and the user must sit in front of it so that his/her face appears in the camera output.

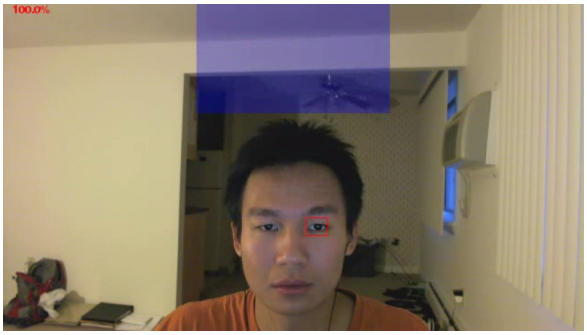


Figure 4. The red box is drawn on the left eye when both eyes are detected. The user can adjust the position and orientation of the face such that the eye detection rate, shown on the top left corner of the screen, is reasonably high in order to achieve good gaze tracking results.

Our gaze tracking program requires the user to install the open-source computer vision library called SimpleCV[9]. SimpleCV is nascent—but growing—Python implementation of OpenCV, which is one of the widely used computer vision libraries. By using Python programming language and SimpleCV, we were able to rapidly prototype and test our ideas for gaze tracking. Another software library that needs to be installed is PyBrain[10], which has implementation of Neural Network and other artificial intelligence modules. Both SimpleCV and PyBrain have simple installation process for all major operating systems. The communities behind these resources are very active and helpful, which is crucial to the success of programming projects.

4. Eye Extraction Model

In order to track the gaze, our program detects the face and the eyes of the user in each video frame. We utilized the real time face detection method of Viola-Jones[22], which is based on detecting the Haar features in a given image and classifying them using cascade classifier architecture. When the program detects the face, it searches for the eyes just within the face frame to achieve computational efficiency. For the eye detection, we use Viola-Jones' approach, but with a different set of Haar-features provided by SimpleCV. We found that the eye detection rate is near perfect--that is, almost always 100 percent--provided that the user's face appears almost full in the camera.

Once the eyes are found, the algorithm selects the left third of the bounding box of the eyes, and extracts the iris within that area. Before extracting the iris, we convert the image into grayscale, use a constant threshold value--empirically found the optimum value to be 230 for grayscale--to transform the image into binary image, and apply the morphological opening operation to it so that the gaps, if any, inside the blob detected are closed. The algorithm ignores the blobs that have area less than 5 pixels.

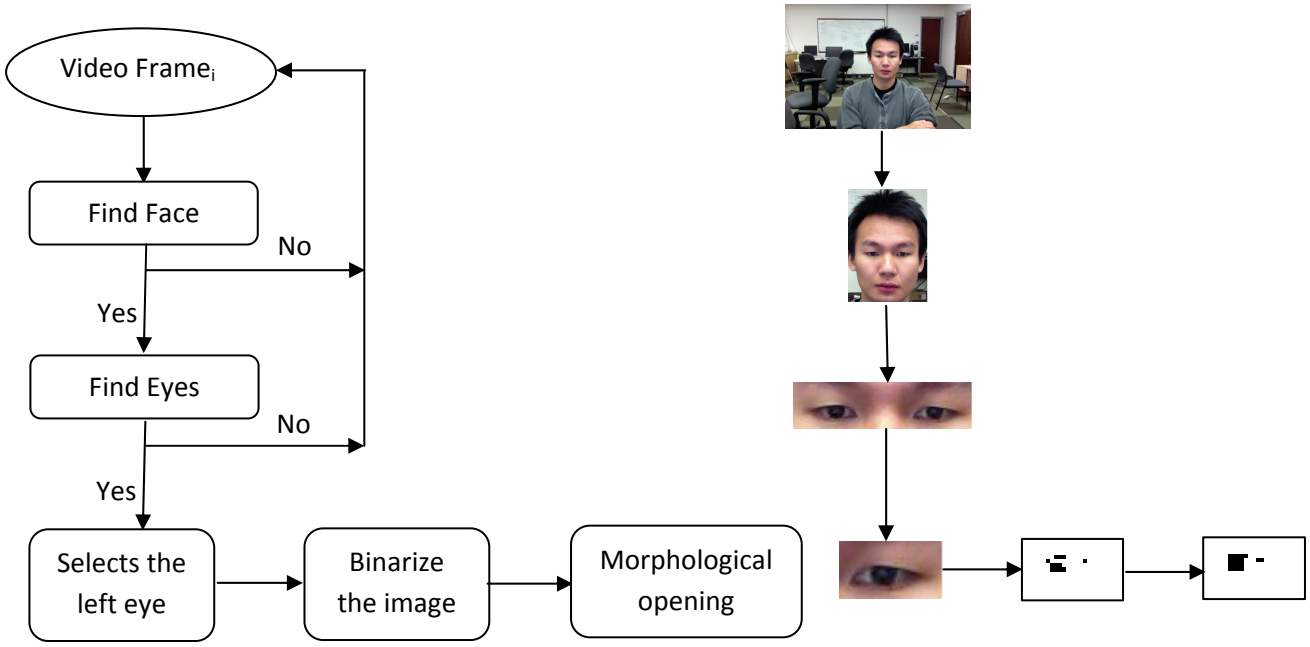
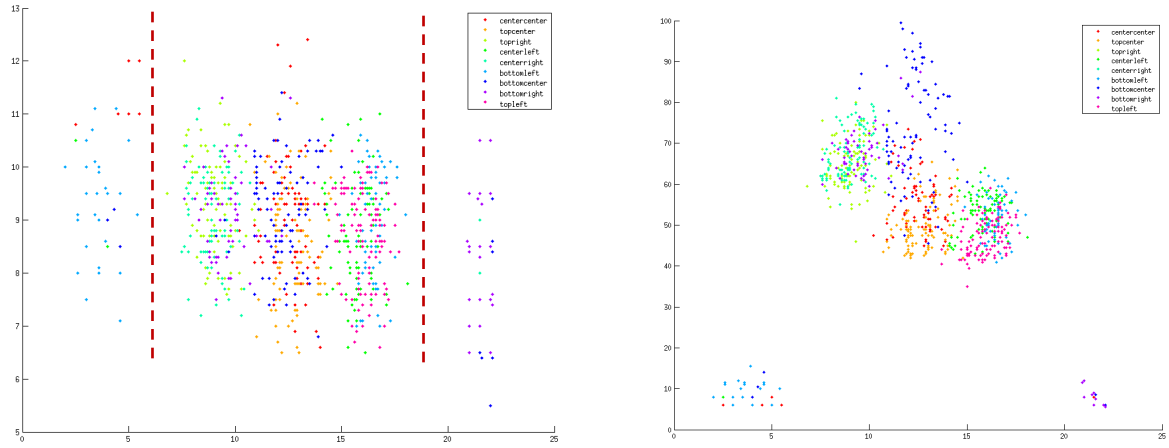


Figure 5. Flow diagram of the iris extraction algorithm and the resulting images are shown side by side.

5. Eye Tracking Methods and Results

Our initial plan was to calculate the centroid and area of the extracted blobs of iris in previous section, and use them to distinguish the gaze points. We collected a few thousands data points for the centroid and the area--so that we had at least 200 data points for each position of eye such as top-left, top-center, top-right, center-left and so on. Analyzing the data of the centroid (x vs. y), we found that although we can find a constant 'x' value to classify the data points into three distinct groups--left, center and right--the data points along 'y' coordinates are linearly inseparable (see figure 5a). We also found that there is no relationship between x-coordinate and the area as can be seen in figure 5b.[1]



a) b)

Figure 5. a) The plot between x vs. y (centroid location) shows that the left, center and right gaze points are linearly separable, whereas those between top, center and bottom are not. **b)** The plot of x (from centroid) vs. area (of the extracted iris) shows no clear boundary between data points for different gaze points. The same is true for "y vs. area", but we cannot provide the plot in this paper due to space constraint.

5.1 Support Vector Machines (SVM)

After we learned that the data is linearly inseparable, we used support vector machines to classify the data. SimpleCV has SVM classifier module for image data, and we used binarized image as input to the classifier. There are different parameters in SVM including kernel type, slack variable (c), gamma (γ), and degree and coefficient for polynomial kernel. The SVM classifier in SimpleCV offers four types of kernel: Radial Basis Functions (RBF), Linear, Polynomial and Sigmoid. We experimented with different kernels and their associate variables—such as gamma and 'c'—to tune the result. The best combination of parameters is found to be Polynomial kernel with degree of five, and coefficient of

[1] In fact, we also analyzed the relationship between the y-coordinates and corresponding area values, and found no relationship as in the case of "x vs. area".

above 30. The rest of the variables are left as default provided by SimpleCV. The confusion matrix from one of many results of SVM classifier, with aforementioned optimal configuration, is shown in figure 6.

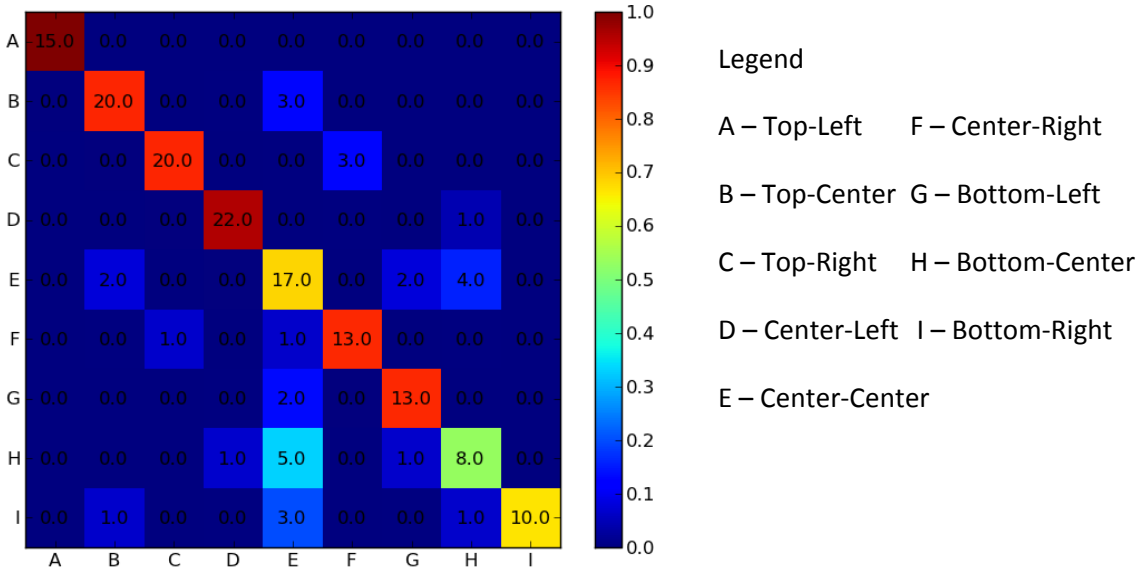


Figure 6. Confusion matrix showing test results of SVM classifier for polynomial kernel with degree of 5 and coefficient of 36. In this particular experiment, we collected ~112 images per gaze point (class), divided them into 80%-20%, trained on the training set of 80% data and tested on the remaining 20%.

The overall accuracy of the run was 65%. We found that the implementation of SVM in SimpleCV uses one-versus-all approach, which might explain why SVM doesn't perform as well as we expected. We hope that extending the SVM classifier to use a one-versus-one classification with plurality voting (that is, winner-take-all) might perform better than the one-versus-all because it might be easier for the classifier to distinguish between two different types of gaze points than it is to compare one type against the ensemble of others. We did not carry out the experiments with this idea because it would require a major implementation overhaul, which deserves a project on its own, to the SimpleCV's SVM classifier. In addition, if we were to use one-versus-one classification, we would have to spend additional computation time for calculating all of the 36 possible combinations—such as top-left vs. top-center, top-left vs. top-right and so on—before we can find the plurality of the vote among them.

5.2 K-Nearest Neighbors (KNN)

The second type of classifier we experimented with is K-nearest neighbor (KNN). SimpleCV has implementation of KNN with various distance functions such as Hamming, Maximal, Manhattan, Euclidean and Normalized. We experimented with all these distance function except Maximal using the RGB and binarized images as features. The best results from KNN experiments is shown in figure 7, with the overall accuracy of about 10%, which is slightly worse than randomly guessing the classes (gaze points). It was not a surprising result given that KNN usually requires more sophisticated feature sets as input as opposed to what we were able to provide: images with RGB or binary values. We hope to

improve the classification result by creating feature vectors such as bag-of-words for classification, but due to the limited time allowed for the project, we didn't explore further for this idea.

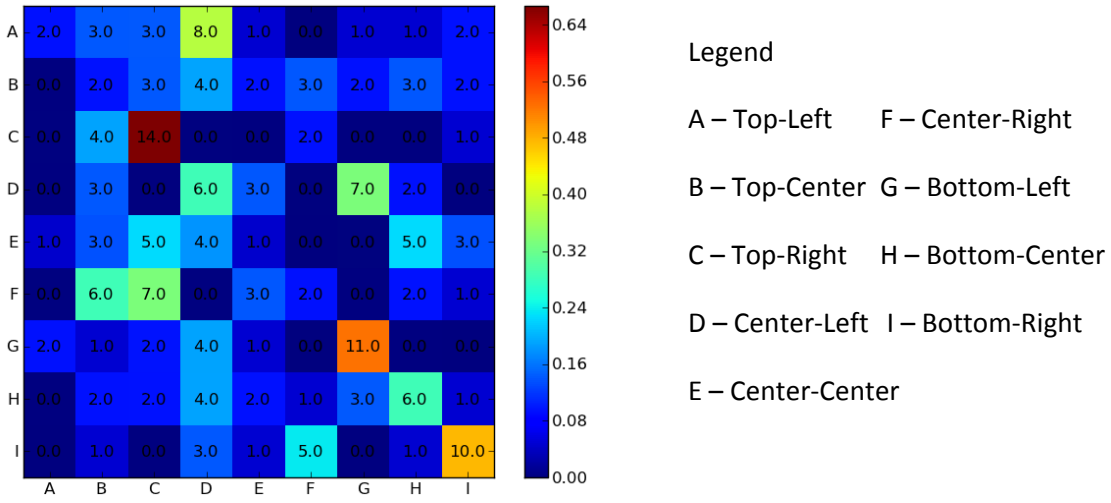


Figure 7. Confusion matrix showing test results of KNN classifier using Euclidean distance as heuristic. In this particular experiment, we collected ~100 images per gaze point (class), divided them into 80%-20%, and obtained overall accuracy of ~10%.

5.3 Feed-forward Neural Network with Backpropagation (FNN)

As a last resort, we classified the gaze point using artificial neural network with backpropagation. We used PyBrain for building and training the neural network and experimented with different number of input and hidden layer nodes as well as two different activation functions--sigmoid and tanh—that are available in PyBrain. In our case, images are of constant size (25x20px) and therefore, we have 500 pixels as our baseline input nodes to the neural network. We tried up to 500 hidden layer nodes, but that proved to be too computationally intensive—it took about two hours to finish training the network. Therefore, we compromised with 40 hidden layer, which is empirically found to train the network consistently above 80% accuracy in 3 minutes for about 80 training epochs. Our final configuration for the neural network is as follows; 500 input nodes, 40 hidden nodes, 9 output nodes, "tanh" as objective function, 1.0 as learning rate, and 0.01 as weight decay. This configuration is empirically found to achieve up to 86% accuracy for training data of over 1000 images (approximately 110 images per gaze point direction) with 80%-20% training-test ratio. We provide the confusion matrix from this experiment in figure 8.

6. Discussion

We observed that our gaze tracking system is susceptible to ambient lighting. If the room lighting is imbalanced—for example, more light is coming from one side of the room than the other—it can severely affect the iris blob extraction algorithm since we are using a constant threshold to binarize the image. Using Hue-Saturation-Value (HSV) or other illuminant invariant color spaces might eliminate this problem.

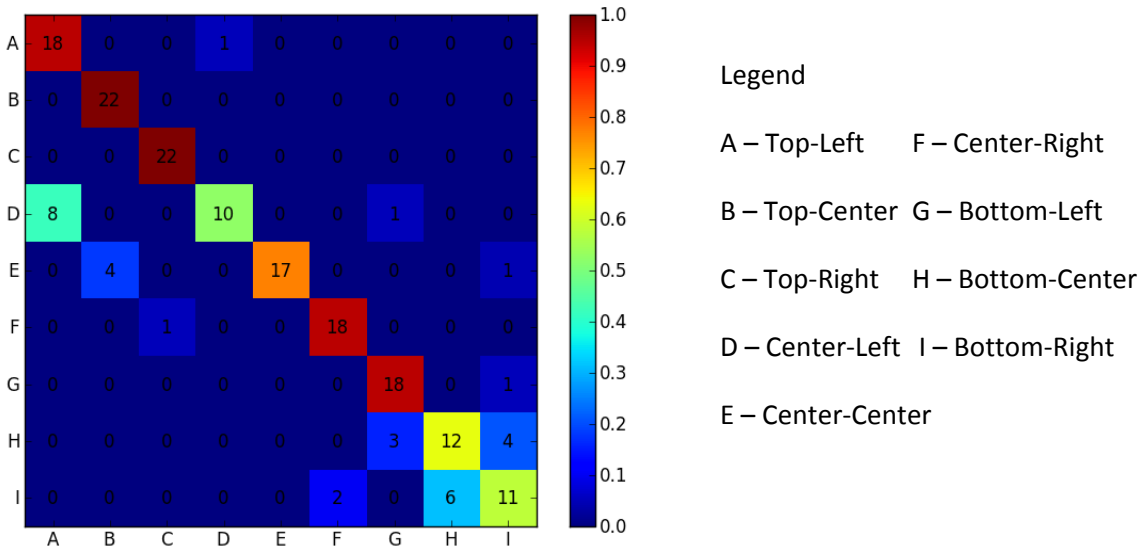


Figure 8. Confusion matrix showing test results of neural network classifier with 500 input nodes, 40 hidden layers and 9 output nodes. The overall accuracy is ~82%.

In addition, we found that in order to get good training samples, the user should be using a monitor that is big enough so that when following the training blobs to the corners and edges, her iris will noticeably move within the eye socket. The effect of the screen size is noticeable when we collected training samples using our laptop, which has 15" monitor, vs. the desktop, which has 20" monitor. We found that the desktop training samples yield better classification rate--up to 86%--than that on the laptop, which averages around 72% accuracy. We believe that this problem could only be overcome when our webcam has resolution high enough to capture the glint from the eye, which will allow us to measure the movement of the eye ball in higher precision.

7. Conclusion and Future Work

In this paper, we present a gaze tracking system that requires inexpensive webcam and very few software dependencies. Our gaze tracker can robustly track the eye gaze, and with proper training samples and lighting conditions, can achieve up to 86% accuracy. Nevertheless, the gaze tracker we developed is not without its share of weaknesses. For one, it has the initial cost of training, which could take as long as 3-4 minutes depending on the computing power of the machine; however, once trained, the user can reuse the trained network at a later time provided that the ambient lighting is not dramatically different from when it was originally trained.

We also plan to reduce the time for training by modifying the system to do online training, in which the user will spend at most one minute of following the training boxes on the screen to build the preliminary neural network. Once the initial training is done, the system will adapt the network by bootstrapping the previously trained one and collecting new samples as the user keeps using the system. This will allow the system to adapt to changes in ambient lighting.

Last but not least, we would like to do more literature research on how to reliably extract the glint or iris blob even under various distortion and head orientation. Currently, our system will have a difficult time detecting the eyes if the user tilts her head by as much as 15 degrees from normal, upright position. Furthermore, it would be extremely useful if we could track the gaze in more finer resolution--that is, track it in smaller areas on the monitor. We hope to continually improve this gaze tracking system as it holds promise for further improvements. We would also release it as open-source on GitHub[23] so that anyone who is interested in testing or participating in the improvement of this system may easily do so at no cost.

Acknowledgements

We would like to thank Professor Jiebo Luo for his insightful guidance throughout the course of the project. This project would not have been possible without SimpleCV, PyBrain and other open-source libraries they are built upon. We would like to thank everyone behind such resourceful and helpful open-source projects.

References

[1] ISCAN - Eye & Target Tracking Instrumentation. 2 May, 2012. URL: http://www.iscaninc.com/

[2] SensoMotoric Instruments. 2 May 2012. URL: http://www.smivision.com/en/gaze-and-eye-tracking-systems

[3] eyetracker™. 2 May 2012. URL: http://www.eyetracker.co.uk/

[4] Eye Tracker Prices. Arrington Research. 2 May 2012. URL: http://www.arringtonresearch.com/prices.html

[5] ITU Gaze Tracker. ITU GazeGroup. 2 May 2012. URL: http://www.gazegroup.org/

[6] Eye/Gaze Tracking. Team 4 of Summer School on Image Processing (SSIP 2009). 2 May 2012. URL: http://www.inf.unideb.hu/~ssip/teams/team4/index.html

[7] Opengazer: open-source gaze tracker for ordinary webcams. 2 May 2012. URL: http://www.inference.phy.cam.ac.uk/opengazer/

[8] Github. 2 May 2012. URL: https://github.com/stefan-k/eyetracker

[9] SimpleCV. The Open Source Framework for Vision. 2 May 2012. URL: http://simplecv.org/

[10] PyBrain. The Python Machine Learning Library. 2 May 2012. URL: http://pybrain.org/

[11] L. Young and D. Sheena. Survey of Eye Movement Recording Methods, Behavior Research Methods and Instrumentation, Vol. 7, No. 5, pp. 397-429, 1975.

[12] Jacob, Robert J K, and Keith S Karn. “Eye tracking in human-computer interaction and usability research: Ready to deliver the promises.” Ed. J Hyona, R Radach, & H Deubel. *Work* 2.3 (2003) : 573-605.

[13] A. Glenstrup and T. Angell-Nielsen, "Eye Controlled Media, Present and Future State," Technical Report, University of Copenhagen, URL: http://www.diku.dk/users/panic/eyegaze/, 1995.

[14] G. Myers, K. Sherman and L. Stark, Eye Monitor, IEEE Computer Magazine, Vol. March, pp. 14-21, 1991.

[15] Y. Ebisawa, Improved Video-Based Eye-Gaze Detection Method, IEEE IMTC '94, Hamamatsu, May, 1998.

[16] T. Hutchison, K. White, Jr., W. Martin, K. Reichert, and L. Frey, Human-Computer Interaction Using Eye-Gaze Input, IEEE Trans. on Systems, Man, and Cybernetics, Vol. 19, No. 6, pp. 1527-1534, 1998.

[17] S. Baluja and D. Pomerleau, Non-Intrusive Gaze Tracking Using Artificial Neural Networks, CMU Technical Report, CMU-CS-94-102, School of Computer Science, Carnegie Mellon University, January, 1994.

[18] B. Hu and M. Qiu, A New Method for Human-Computer Interaction by using Eye Gaze. *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, October, 1994.

[19] M. Chau and M. Betke. Real time eye tracking and blink detection with USB cameras. Boston University Technical Report, 2005-12, Computer Science Department, Boston University. May 12, 2005.

[20] T. Ohno, N. Mukawa and A. Yoshikawa. FreeGaze: A gaze tracking system for everyday gaze interaction. In *Proceedings of Eye Tracking Research and Applications (ETRA2002)*, 125-132.

[21] Stiefelhagen, R., Yang, J. and Waibel, A. Tracking eyes and monitoring eye gaze. *Proceedings of the Workshop on Perceptual User Interfaces* (1997), pp. 98-100

[22] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, Vol. 1 (2001)

[23] GitHub - Social Coding. URL: https://github.com/