CS 231A Final Project
Real-time Webcam Gaze-tracking

Wen Hao Lui
whlui@stanford.edu

Abstract

Gaze-tracking is a useful technology that opens new avenues for HCI and research into human behavioral patterns. However, the image quality of current web cameras and the need for real-time image processing makes specialized hardware the norm in commercial offerings. We present a machine-learning based approach using well-engineered visual features to enable an accurate estimate of the user's gaze position using normal webcam hardware, without the need for mounting on the user. Unlike other webcam-based approaches, we also model the user's head orientation to allow the user greater flexibility in viewing postures. Our successful experiment, with a median error of 3.22° in determining the user's gaze position, validates our approach.

1. Introduction

Eye-tracking is a valuable capability that has versatile applications, such as an input to a device or a measurable metric for research. The strong eye-mind hypothesis [5] postulates that the amount of time a person spends thinking about an object is proportional to the time spent looking at it. Based on this premise, several companies like EyeTrackShop, YouEye and GazeHawk are offering gaze-tracking services to allow product creators to see if users are looking at the correct elements of the product, as opposed to being distracted by a side icon for example. Besides usability testing, gaze tracking can also be used for market research, such as determining the products that grab attention in a shopping aisle, or identifying the most eye-catching features of a billboard advertisement. Researchers also use eye-tracking to study human tasks like reading, web surfing, and driving [4] in more detail. Another interesting application is in HCI [6, 8], where the user can use their eyes to direct input to an electronic device. Besides the convenience afforded for some tasks, this also allows physically impaired people to have an alternative input system.

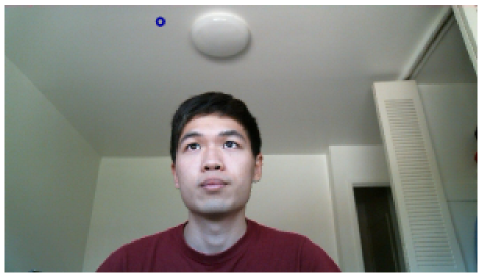

Figure 1. **Illustration of problem.** Given an image of the user captured by the webcam, we want to identify the gaze co-ordinates of the user (highlighted with the blue circle).

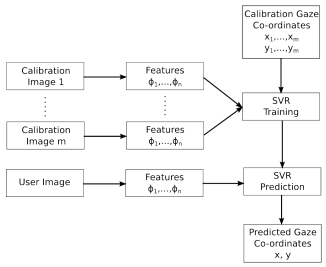
Calibration Gaze Co-ordinates $x_1,\ldots,x_m$ $y_1,\ldots,y_m$
Calibration Image 1
Features $\phi_1,\ldots,\phi_n$
SVR Training
Calibration Image m
Features $\phi_1,\ldots,\phi_n$
User Image
Features $\phi_1,\ldots,\phi_n$
SVR Prediction
Predicted Gaze Co-ordinates x, y

Figure 2. **High-level overview of our approach.** We collect calibration images tagged with the gaze co-ordinates, then train a SVR model on the extracted features from the images. The SVR model is then used to predict a new set of gaze co-ordinate based on extracted features from a new input image.

1.1. Challenges

There are numerous challenges involved in gaze tracking, as explained by Kumar et al. [7]. The use of web cameras dictate that the resolution of our video frames will be lower. Since the user's eyes take up a small fraction of the captured image, the actual image size that we work with is on the order of 60 by 30 pixels for the whole eye. As the gaze shifts from the left edge to the right edge of the screen, the pupil moves about 13 pixels at most from its original position. A direct mapping will give a very coarse position estimate, further worsened by noise; hence we need additional features to further fine-tune the output.

Eye movement is also naturally jerkey, and saccades are a well-studied phenomenon in humans and other animals. We aim to reduce the effect of this erratic behavior by applying a filter to smooth out the movement.

Given the predicted gaze position P for a frame of the video stream,we apply an exponential moving average filter or another smoothing algorithm in order to reduce the effects of saccades (rapid eye movements, usually done subconsciously) and noise from measurement.

1.2. Our work

We choose to work on a webcam platform that is already present in most user environments. The main challenge comes with the comparatively low image resolution and wide field of view - this means that only a small number of pixels are used to represent the eye, and makes determining visual features difficult and error-prone.

We tackle the usual formulation of the problem - given an image of the user, as shown in Figure 1, we want to determine the location on the screen that the user is looking at. Our approach uses support vector regression (SVR) on robust features to obtain reliable estimates of the eye position. As shown in Figure 2, using calibration images matched with the corresponding gaze co-ordinates of the user, we extract the relevant features and train a linear SVR model. This SVR model is then used to predict the gaze co-ordinates of future input images.

2. Previous Work

Traditionally, eye-tracking has been carried out with specialized equipment. Some invasive tools used include special contact lens and magnetic search coils [1]. As video-recording and data-processing capabilities improved, there has been greater interest in non-invasive optical methods that can yield acceptably accurate results. Companies like Tobii have developed specialized cameras that give accurate readings of the user's gaze point, allowing for direct fine-grained control [9]. There has also been open-source development of eye-tracking software such as Opengazer [11] and Gaze Tracker [2] that allows home users to turn their webcams into functional eye-trackers.

2.1. Main Contribution

This project will follow in the spirit of these open-source projects and aim to create a working eye-tracking program using a webcam. We plan to use easily available hardware like web cameras, instead of specialized cameras, so as to keep costs low and make the potential user base as large as possible. We also want to avoid using any equipment mounted on the body, which would bring inconvenience to the user and again require specialized equipment.

We take Opengazer's accuracy as a rough benchmark for expected performance. It has an error of about 1.5-inch on a 15-inch screen. This translates to an angular error of about 4°. There were no other publicly available real-time webcam-based gaze-tracking software available for comparison (that did not require the user wearing a camera). Although there are inherent limitations due to the poor resolution of web cameras, we discuss how we plan to approach this lower error threshold in section 3.4.

Our learning model also improves on both commercial and open-source gaze-tracking offerings, which typically require the user to hold their head in a still position. By modeling the orientation of the user's head, we can maintain accurate predictions of the gaze co-ordinates despite the user shifting his or her head. This allows greater comfort in using the gaze-tracking software.

3. Approach

3.1. Overview

We collect training data through calibration, where the user sits still and looks at various pre-determined points on the screen while the program records the eye features. We then use computer vision techniques to extract key features from the calibration images. An SVR model is trained, based on these extracted features and their corresponding gaze co-ordinates. The model can then be used to make predictions of gaze co-ordinates given the extracted features of a new image.

3.2. Problem Formulation

Given a webcam stream containing a user, we want to identify the point on the screen that the user is looking at. Parameters like screen size, position of camera relative to the screen, screen resolution are assumed to be given. Additionally, we will make use of existing features in libraries like OpenCV for facial detection and eye recognition to cut down on the amount of pre-processing work needed. For subsequent discussions, we assume that the necessary pre-processing has been done so that the two eye positions are known, and the positions of the eyes in the picture are also known.

For each image Q of the web stream, we want to output a screen co-ordinate $P = (x, y)$ that is as close as possible to the true gaze point $\hat{P}$. The error is then the pixel distance from the predicted co-ordinate to the actual co-ordinate, $\epsilon = ||P - \hat{P}||$. This error distance can then be translated to an angular error with the given distance of the face from the camera and the screen resolution.

3.3. Features

We choose to work with a small number of features in order to simplify the model and allow it to better generalize to unseen examples. This is needed because our low quantity of training data (arising solely from calibration images) makes our model very likely to overfit as we increase the number of features.

We present our features and the methods used for feature extraction here. The methods chosen needed to fit the criteria of precision (low variance) and speed. We need precision due to the sensitivity of the gaze co-ordinates to slight changes in feature value, while the speed requirement was imposed to enable real-time feedback to the user. After experimenting with several detection methods (RANSAC, Hough transforms, sliding windows, shape contexts, SIFT feature matching), we found that the most reliable method was the generalized Hough transform [3]. By imposing certain constraints on the search space, we are able to perform the required transforms in real-time. The list of features is summarized in Table 1.

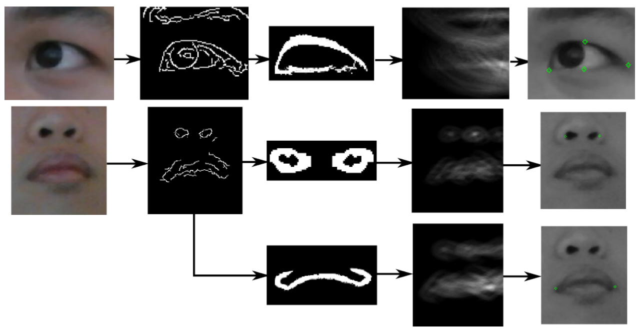
Figure 3. **Generalized Hough transform for eyes, nose and mouth.** Image descriptions from left to right: original image, Canny edge detection applied, shape template used, Hough voting pattern, identified keypoints based on most popular location vote. We apply Canny edge detection to the given images, then use the detected edges and a template to vote for the position of the eye/nose, taking the highest-scoring pixel as the correct position. The Hough voting pattern uses as the origin the right corner of the eye, the right nostril, and the right corner of the mouth.

3.3.1 Face position in image

We want to localize the face in the image so that we know the reference frame the user is looking from. In addition, it gives us a smaller search space for our subsequent features, saving computation time. This feature was calculated using Haar cascades [10], and is a method already implemented in OpenCV.

3.3.2 Eye position in face

Using Haar cascades only gives us a bounding box for the eye position, which is not accurate enough for our purposes. We use a generalized Hough transform, as shown in Figure 6, to estimate the position of the eye on a pixel level. We apply a Gaussian filter to remove noises, then use a Canny edge detector to identify the edges that would comprise our shape of interest. For each eye, we have a shape template based on the average shape profile of about 10 eyes. Each pixel in the detected edges then votes for the position of the eye, casting one vote from the perspective of each pixel in the shape template. That is, if the origin of the eye is o, then from the perspective p of the pixel in the shape template, the pixel in location q in the original image would cast a vote for the pixel in location $q + (o - p)$. Tallying up the votes, the pixel with the highest count would be the predicted eye position.

3.3.3 Pupil position in eye

We use Hough circles to detect the pupil positions, specifying a minimum and maximum radius in order to limit the search space. The gradient-based approach in OpenCV did not yield sufficiently consistent circle centers, so we implemented our own approach based on Canny edge detectors (Figure 4).

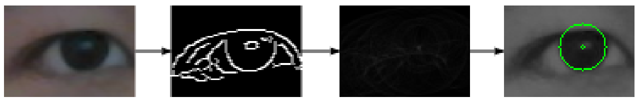
Figure 4. **Hough circle transform for pupil detection.** We impose a tight constraint on the minimum and maximum radii to keep the search space small.

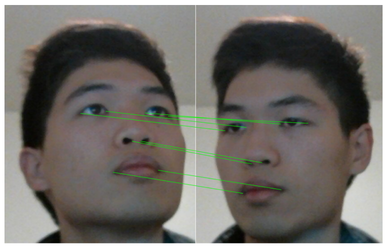
Figure 5. **Matching keypoints for transformation matrix calculation.** We use four corresponding points (one each from the eyes, nose and mouth) to specify the transformation matrix from one image to the other.

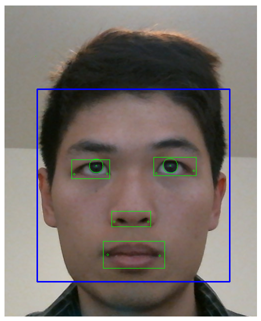
Figure 6. **Identified key features in the image.** The blue rectangle indicates the bounding box found by the Haar cascade for face detection. Eye, nose, and mouth positions were found using generalized Hough transforms. We found pupil positions using Hough circle transforms.

3.3.4 Head orientation

We use additional generalized Hough transforms to determine the nose and mouth positions. In addition to the two eye positions, we now have four key points that we can use to generate point correspondences (see Figure 5) between any two images of the face. We can use this to calculate the transformation matrix that converts points in one image to another, and from the coefficients of the matrix we can deduce the orientation angles of the head.

3.4. Learning

Given the features in Table 1, we can perform linear regression with an SVM to map the relationship between

Feature	Description
ϕ_0	Constant bias term
ϕ_1	Face position in image
ϕ_2	Eye position in face
ϕ_3	Pupil position in eye
ϕ_4	Head orientation

Table 1. List of features for position estimation.

features and the gaze co-ordinates. We can use a linear kernel due to the geometry of the problem - the gaze co-ordinate varies linearly with the measured feature. In the case of some non-linear features (such as angles), we do a pre-computation by taking the arcsine of the angle with respect to the zero position. Using a linear kernel also helps with learning, because of the low amount of calibration images we can obtain. A higher-dimensional kernel will suffer from the curse of dimensionality and very likely overfit the training data.

We train two separate SVMs, one for the x-axis and one for the y-axis. The optimization problem for the SVM becomes:

$$\min_{w_x} \frac{1}{2} w_x^T w_x + C \sum_i \max(|x_i - w_x^T \phi(Q)| - \epsilon, 0)$$

Where $\phi(Q)$ is a vector generated by the features described earlier, C is the weight given to errors, and ϵ is the threshold distance within which a prediction is labeled correct. A similar formulation holds for the y variables. The ϵ term allows the regression to focus on the mis-predicted examples as support vectors instead. We also use the absolute deviation instead of squared deviation to make the regression more robust and less easily skewed by outliers (due to the non-Gaussian distribution of error).

The predictor for the gaze co-ordinates is then:

$$(x, y) = (w_x^T \phi(Q), w_y^T \phi(Q))$$

4. Experiments

We collected 60 images for training our model. Each picture is taken when the user is looking at a randomly-generated co-ordinate on the screen, and subsequently tagged with the same co-ordinate. Using this training set of 60 images with labeled gaze co-ordinates, we trained 60 different models using the leave-one-out cross-validation approach in order to maximize the use of limited training data. We report the aggregated error for the trained models on their respective unseen test images in Figure 7. The median error is **148.3 pixels** and the mean error is 240.3 pixels. The experiment was conducted on a 22-inch screen with 1920x1080 resolution, viewed from a distance of 25 inches. This corresponds to a median angle error of **3.22°** and a mean error angle of 5.22°. As can be seen from the error histogram, the error follows an exponential distribution that makes outliers skew the mean error significantly. We use the median error as a more robust gauge of the performance of our system.

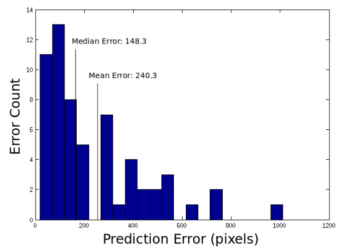

Figure 7. **Histogram of Testing Error.** The error over the 60 training examples is computed using leave-one-out cross-validation. The mean error is 240.3 pixels and the median error is 148.3 pixels

5. Conclusion

Using a few well-engineered features and a simple regression method, we have managed to the predict the user's gaze co-ordinates with a median error of 3.22°. This compares favorably with other commercial and open-source offerings. As a comparison, most webcam-based approaches have an error in the 2.5° to 5° range, while the median error with specialized hardware like the Tobii Eye Tracker is around 0.5°.

The major shortcoming with current eye-tracking software is that the user is required to sit still in order for the calibrated model to be accurate. This makes gaze-tracking technology uncomfortable for medium to long-duration use. One future extension to this work is to incorporate more of the user pose data into the model, so that the model can still make accurate predictions even if the user's pose changes. We currently use the head orientation and position, but have yet to include the distance of the user from the webcam. The user distance makes a significant impact on the gaze point, with the error scaling linearly. Incorporating this variable into our model will further improve its accuracy and also afford greater pose flexibility to the user.

References

[1] David A. *Robinson: A method of measuring eye movement using a scleral search coil in a magnetic field, IEEE Transactions on Bio-Medical Electronics*. October 1963.

[2] San Agustin, Skovsgaard J., Mollenbach H., Barret E., Tall M., Hansen M., D. W., and J. P. Hansen. Evaluation of a low-cost open-source gaze tracker. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications (Austin, Texas, March 22-24*, pages 77–80. 2010. URL `http://doi.acm.org/10.1145/1743666.1743685`.

[3] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[4] A. S. Cohen. *Informationsaufnahme beim Befahren von Kurven, Psychologie für die Praxis 2/83*. 1983.

[5] M. A. Just and P. A. Carpenter. Eye fixations and cognitive processes. *Cognitive Psychology*, 8:441–480, 1976.

[6] M. Kumar. Reducing the cost of eye tracking systems. Technical Report CSTR 2006-08, Stanford University, Stanford, April 2006. URL `http://hci.stanford.edu/cstr/reports/2006-08.pdf`.

[7] Manu Kumar, Jeff Klingner, Rohan Puranik, Terry Winograd, and Andreas Paepcke. Improving the accuracy of gaze input for interaction. In *ETRA '08: Proceedings of the 2008 symposium on Eye tracking research & applications*, pages 65–68, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-982-1. doi: 10.1145/1344471.1344488.

[8] Alex Poole and Linden J. Ball. Eye tracking in human-computer interaction and usability research: Current status and future prospects. 2010.

[9] S. Stellmach and R. Dachselt. *Gaze-supported Interaction (Demo)*. Presented at the Mensch & Computer Workshopband, 2012.

[10] Paul Viola and Michael Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Conference on Computer Vision and Pattern Recognition (CVPR), 2001.

[11] P. Zielinski. *Opengazer: open-source gaze tracker for ordinary webcams (software), Samsung and The Gatsby Charitable Foundation*. 2008. URL `http://www.inference.phy.cam.ac.uk/opengazer/`.