Eye Gaze Tracking with Free Head Movements using a single camera

Ba Linh Nguyen
CHArt EA 4004-UMS CNRS 2809
École Pratique des Hautes Études
41 rue Gay Lussac
F - 75005 Paris
linhnb@gmail.com

Youssef Chahir
GREYC - UMR CNRS 6072
Université de Caen
Campus Côte de Nacre
F - 14032 Caen Cedex
chahir@info.unicaen.fr

Michèle Molina
Laboratoire PALM JE 2528
Université de Caen
F - 14032 Caen Cedex
michele.molina@unicaen.fr

Charles Tijus
CHArt EA 4004-UMS CNRS 2809
École Pratique des Hautes Études
Université Paris 8
F - 93200 Saint-Denis
tijus@univ-paris8.fr

François Jouen
CHArt EA 4004-UMS CNRS 2809
École Pratique des Hautes Études
41 rue Gay Lussac
F - 75005 Paris
francois.jouen@ephe.sorbonne.fr

ABSTRACT

The problem of eye gaze tracking has been researched and developed for a long time. The most difficult problem in the non-intrusive system of eye gaze tracking is the problem of head movements. Some of existing methods have to use two cameras and an active infrared (IR) illumination to solve this problem. Otherwise, with a single camera, the user has to hold the head uncomfortably still when performing a session of eye gaze tracking. If the head of the user moves away from original position, the accuracy of these eye gaze-tracking systems drops dramatically. In this paper, we propose a solution using Gaussian Processes for eye gaze tracking that allows free head movements with a single camera.

Keywords

eye detecting, eye gaze tracking, Gaussian process

1. INTRODUCTION

In recent years, many non-intrusive systems of eye gaze tracking have been researched and developed. Most of them use infra red (IR) lighting to illuminate the pupil then extract the eye orientation by exploiting geometrical properties of the IR spotlights reflections [11], [6], [16], [13], [12], [8], [2] and use two cameras for calibration [11], [2]. Yet these non-intrusive methods for tracking eye gaze in real time have not solved the problem of the head movement. Users have to hold the head uncomfortably still when performing an eye gaze tracking session. If the head of user moves away from original position where the user performs the eye gaze calibration, the accuracy of these eye gaze tracking systems drops dramatically.

To solve the problem of the head movement, Zhiwei Zhu and Qiang Ji have used two cameras for calibration, and an active infrared (IR) illumination to obtain accurate eye location through corneal reflection [17]. This system allows user move the head freely when tracking the eye gaze. But to use this system, user has to install two cameras and an active infrared illumination. For this drawback, this method is till not used widely in practice.

In this paper, we propose a non-intrusive solution using Gaussian Processes for eye gaze tracking that allow tree head movements with only a single camera. Firstly, we use the method of detecting object base on the Haar-like features [15], [7] to detect the eye on the face of the user. This method trains classifiers with a few thousands of sample view images of object and construct a cascade of classifiers to detect object rapidly. Secondly, we track the eye by tracking the corners on the face of user. We find corners on the eye, the nose and the mouth of user and we use the algorithm of feature tracker of Lucas Kanade [3] to track corners, and then we use the Outliers detection method [9] to detect the corner that is lost of tracking and recover it. Lastly, we track the eye gaze on the screen. We have to find the functional relationship between the user's eye and the point on the screen where the user is looking at. We do not use neural network to train this function, but we will estimate the distribution of this function. We use Gaussian process to calculate the mean and the covariance of this function and use this function to make predictions for the new inputs that we have not seen in the training data. With this method, the user has to perform a calibration procedure to obtain a gaze mapping function with different visual angles of the head. So that, after the calibration, the user can move his head freely in front of camera, which will make the communications between the computer and user naturally. The user can perform eye gaze tracking everywhere with any simple camera (camera built-into a computer portable or camera connected to computer by a USB port).

The remainder of this paper is organized as follows. Section 2 details the detection of the eye using Haar-like features, section 3 then outlines the method for detecting and tracking corners and the method for detecting the errors of tracking to correct them while section 4 describes the using of the Gaussian process to predict the gaze. Section 5 presents experimental results, section 6 gives our conclusions and further research.

2. DETECTING EYE

To make the prediction of the eye gaze of a user, we have to detect the eye of the user on camera. Here we use the rapid object detection scheme based on a boosted cascade of simple Haarlike feature to detect the eye. This method has been initially proposed by Paul Viola [15] and improved by Rainer Lienhart [7]. First, the classifiers are trained with thousands of positive and negative images (images of object and images non-object) based on simple features (called Haar-like feature [15]). There are a large number of features in a sub-window of image 24x24 pixels (117,941 features) [7], a number far larger than the number of pixels. The algorithm of training is AdaBoost [15], which trains classifiers and in the same time select a small set of features that are the best separates the positive and negative examples. After training, only a small set of features are selected and these features can be combined to form an effective classifier. After the classifiers are trained, a cascade of classifiers is constructed to increase the detection performance while radically reducing computation time. The cascade of classifiers can be constructed to reject many of the negative sub-windows while detecting almost all possible instances. Simple classifiers are used to reject the majority of sub-windows before more complex classifiers are called to reduce the time computation and to achieve low false positive rates.

In our application we use the classifier which is trained by OpenCV Swiki with 7000 positive samples for the eye at the website http://alereimondo.no-ip.org/OpenCV. The result of this classifier is good for detecting the eye on the image and detecting the eye on the camera in real time. The result of detecting eye is shown in figure 1. But in fact, if we use this classifier to tracking the eye in the real time, it will be very heavy, because it has to search the eye in all frames captured by camera in real time. So we use this classifier to detect the eye in the first frame and then use another method to track the eye for all the following frames. We will present this method in the next section.

3. TRACKING EYE

To track the eye moving in real time, we track the corners on two eyes then recovering the eye in each frame. To have a good tracking, we have to track others corners on the face such as corners on the nose and the mouth of the user to know all the corners are always on tracking or lost of tracking. We use the Outliers detection method to detect the corners that are lost of tracking and recover them.

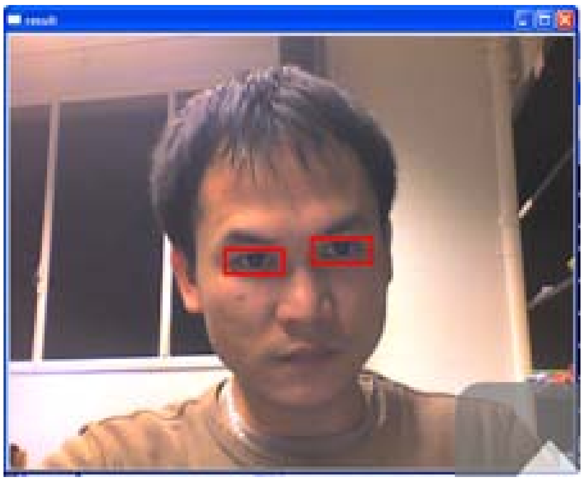

Figure 1: Detecting the eye.

3.1 Searching the corners

In this section we will detect the corners on the eyes, nose and mouth. We use the function CvGoodFeatureToTrack in OpenCV to find two strongest corners in the tail of each eye in the region of each eye in the image that we have detected in the previous section. This function finds corners with big eigenvalues in the image. The function first calculates the minimal eigenvalue for every source image pixel using Harris operator [4], then it performs non-maxima suppression (only local maxima in 3x3 neighborhoods remain). The next step is rejecting the corners with the minimal eigenvalue less than specifies minimal accepted quality of the image corners. Finally, the function ensures that all the found corners are distanced enough one from another by considering the corners (the strongest corners are considered first) and checking that the distance between the newly considered feature and the features considered earlier is larger than specifies minimum possible distance between returned corners.

To find the corners on the nose and the lips, we still use the method CvGoodFeatureToTrack to find the corners in the region prediction below the eye. The result is shown in figure 2.

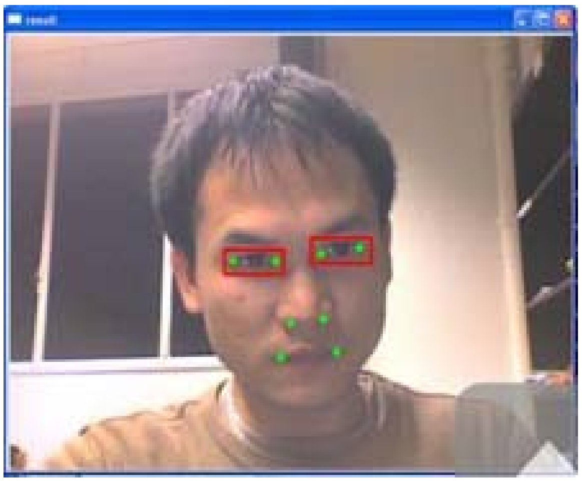

Figure 2: Searching the corners.

3.2 Tracking the corners

To track the eye we use the tracking algorithm Lucas Kanade [3] which is very effectively for tracking objects in real time. After detecting the corners in previous section, we use the method cvCalcOpticalFlowPyrLK in OpenCV to track these points. This method in OpenCV calculates optical flow for a sparse feature set using iterative Lucas-Kanade method in pyramids. The detail of this algorithm is described in [3]. The result of this tracking method is excellent in real time. However, in the process of detecting the eye gaze, when user moves his head a lot, sometimes it produces the errors (the corners are lost of tracking). So we have to detect these errors and recover them.

3.3 Detecting and recovering the tracking errors

We use the Outliers detection method to detect the corners that are lost of tracking. An Outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism [5]. It is a data object that does not comply with the general behavior of the data, it can be considered as noise or exception, which is quite useful in rare events analysis. We use the Distance Based Approach to detect outliers [9]. A popular method of identifying outliers is by examining the distance to an example's nearest neighbors. In this approach, one looks at the local neighborhood of points for an example typically defined by the K nearest examples (also known as neighbors). If the neighboring points are relatively close, then the example is considered normal; if the neighboring points are far away, then the example is considered unusual. The advantages of Distance-Based outliers are that, no explicit distribution needs to be defined to determine unusualness, and it can be applied to any feature space for which we can define a distance measure.

In our problem, the dataset is all the transitions of all corners from one frame to another frame. The outlier is the transition of the corner which have different movements from others corners. That means, if the distance of the corner transition is greater than a specified distance, that transition is outlier and that corner is lost of tracking

To recover the corner that is lost of tracking, we take the sum of the old positions of the corner and the average of the transition of all others corners. The task of detecting the corner lost of tracking and recovering its position is performed in each frame, so that all the corners are being tracked perfectly in all the time.

Now we want to know where the eye looks at on the screen, in the next section we will review the use of Gaussian process to detect the eye gaze, a more detail can be found in [10], the result of this section is the input for detecting the eye gaze in the next section.

4. PREDICTING GAZE USING GAUSSIAN PROCESSES

We have a dataset (training data) which includes the input $\mathbf{x}$ as the image of the eye (32×16), and the output y as the point on the screen where a person is looking at, y have two values (t_x, t_y), we will use these two values independently in the predicting of the eye gaze. We call this dataset is $\mathcal{D}$: $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, ..., n\}$ where $\mathbf{x}$ denotes an input vector (covariates) of dimension D and y denotes a scalar output or target (dependent variable), n is the number of observations. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix X, and the targets are collected in the vector $\mathbf{y}$, so we can write $\mathcal{D} = (X, \mathbf{y})$. In the regression setting the targets are real values t_x or t_y. We are interested in making inferences about the relationship between inputs and targets (Figure 3). A wide variety of methods have been

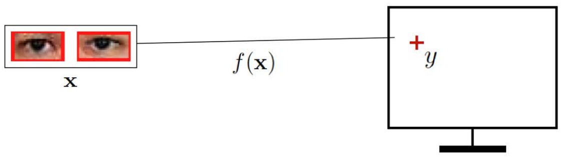

Figure 3: Mapping input-output.

proposed to deal with this supervised learning problem. In common, we use a large set of training data and use neural networks to train the function, but it is difficult to have a good set of training data with all of possible inputs, we use the Gaussian process for this reason. The properties of the function at a finite number of point inferences in the Gaussian process will give us the same answer if we take all other infinitely points. We wish to make predictions for the new inputs $\mathbf{x}_*$ that we have not seen in the training set, we need to move from the finite training data $\mathcal{D}$ to a function f that makes predictions for all possible input value.

The main idea of this method is calculate the predictive distribution for $f_* \equiv f(\mathbf{x}_*)$ at $\mathbf{x}_*$: $p(f_* | \mathbf{x}_*, \mathcal{D})$ and we use Gaussian process to calculate this predictive distribution. A Gaussian process is completely specified by its mean function and covariance function. We will estimate the mean function and the covariance function of f_* with each new $\mathbf{x}_*$.

With all the values in data training $\mathbf{x}_1, ..., \mathbf{x}_n$ we may draw samples $f(\mathbf{x}_1), ..., f(\mathbf{x}_n)$ rom the prior distribution over the functions specified by a particular Gaussian process with mean zero and covariance function K.

$$f(\mathbf{x}_1), ..., f(\mathbf{x}_n) \sim \mathcal{N}(\mathbf{0}, K) \tag{1}$$

where

$$\begin{aligned} K_{p,q} &= \operatorname{cov}(f(\mathbf{x}_p) f(\mathbf{x}_q)) - k(\mathbf{x}_p, \mathbf{x}_q) \\ &= \exp(-\frac{1}{2} |\mathbf{x}_p - \mathbf{x}_q|^2) \end{aligned} \tag{2}$$

The specification of the covariance function implies a distribution over functions. We chose a number of input points, X_* and write out the corresponding covariance matrix using (2) element wise. Then we generate a random Gaussian vector with this covariance matrix

$$\mathbf{f}_* \sim \mathcal{N}(\mathbf{0}, K(X_*, X_*)) \tag{3}$$

The generation of multivariate Gaussian samples is described in [14]. Take a look on the definition of the Gaussian process:

A Gaussian process *is a collection of random variables, any finite number of which has a joint Gaussian distribution.*

Initially, we consider the simple special case where the observations are noise free that is we know $\{(\mathbf{x}_i, f_i) | i = 1, ..., n\}$. Base on the definition of the Gaussian process, we can write the joint distribution of the training outputs,

$\mathbf{f}$, and the test outputs $\mathbf{f}_*$ according to the prior is

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \quad (4)$$

If there are n training points and n_* test points then $K(X,X_*)$ denotes the $n \times n_*$ matrix of the covariance evaluated at all pairs of training and test points, and similarity for the other entries $K(X,X), K(X_*,X_*)$ and $K(X_*,X)$. To get the posterior distribution over functions we need to restrict this joint prior distribution to contain only those functions that agree with the observed data points. Corresponding to conditioning the joint Gaussian prior distribution on the observations to give [14].

$$\begin{aligned} \mathbf{f}_*|X_*, X, \mathbf{f} \ \sim \mathcal{N}\big(&K(X_*,X)K(X,X)^{-1}\mathbf{f}, \\ &K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)\big). \end{aligned} \quad (5)$$

Function values $\mathbf{f}_*$ (corresponding to test inputs X_*) can be sampled from the joint posterior distribution by evaluating the mean and covariance matrix from (5) and generating samples according to the method described in [14].

Return to our problem, we assumed that the observed values y differ from the function values $f(\mathbf{x})$ by additive noise ε, $y = f(\mathbf{x}) + \varepsilon$, and we further assume that this noise follows an independent, identically distributed Gaussian distribution with zero mean and variance σ_n^2

$$\varepsilon \sim \mathcal{N}(0, \sigma_n^2) \quad (6)$$

And the prior on the noisy observations becomes

$$\begin{aligned} \text{cov}(y_p, y_q) &= k(\mathbf{x}_p, \mathbf{y}_q) + \sigma_n^2 \delta_{pq} \\ \text{or} \quad \text{cov}(\mathbf{y}) &= K(X,X) + \sigma_n^2 I, \end{aligned} \quad (7)$$

where δ_{pq} is a Kronecker delta which is one if $p = q$ and zero otherwise. It follows from the independence assumption about the noise, that diagonal matrix is added, in comparison to the noise free case, (2), Introducing the noise term in (4) we can write the joint distribution of the observed target values and the function values at the test locations under the prior as

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \quad (8)$$

Deriving the conditional distribution corresponding to (5) we arrive at the key predictive equations for Gaussian process regression

$$\mathbf{f}_*|X, \mathbf{y}, X_* \ \sim \mathcal{N}\big(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)\big), \quad \text{where} \quad (9)$$

$$\bar{\mathbf{f}}_* \triangleq \mathbb{E}[\mathbf{f}_*|X, \mathbf{y}, X_*] = K(X_*,X)[K(X,X) + \sigma_n^2 I]^{-1}\mathbf{y}, \quad (10)$$

$$\text{cov}(\mathbf{f}_*) = K(X_*,X_*) - K(X_*,X)[K(X,X) + \sigma_n^2 I]^{-1}K(X,X_*). \quad (11)$$

We use a compact form of the notation setting $K = K(X,X)$ and $K_* = K(X,X_*)$. In the case that there is only one test point $\mathbf{x}_*$ we write $\mathbf{k}(\mathbf{x}_*) = \mathbf{k}_*$ to denote the vector of covariance between the test point and the n training points. Using this compact notation and for a single test point $\mathbf{x}_*$, equations 10 and 11 reduce to

$$\bar{f}_* = \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{y}, \quad (12)$$

$$\mathbb{V}[f_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1}\mathbf{k}_*. \quad (13)$$

```
input: $X$ (inputs), $\mathbf{y}$ (targets), $k$ (covariance function), $\sigma_n^2$ (noise level), $\mathbf{x}_*$ (test input)
$L := \text{cholesky}(K + \sigma_n^2 I)$
$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$        } predictive mean (12)
$\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$
$\mathbf{v} := L \backslash \mathbf{k}_*$                                  } predictive mean (13)
$\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$
return: $\bar{f}_*$(mean), $\mathbb{V}[f_*]$(variance)
```

Table 1: The algorithm of prediction.

So we have the implementation of Gaussian process regression for prediction of mean and variance in 1 [14]. The algorithm uses Cholesky decomposition, instead of directly inverting the matrix, since it is faster and more stable.

5. EXPERIMENTAL RESULTS

Before make a session of eye gaze tracking, the user has to perform a procedure of calibration to make the training set. User sits in front of the screen and takes a look to sixteen points which appear consecutively. Each time when a point appears, the camera records his eye and makes a pair of data training (the image of user's eye and the position of the point correspondent on the screen where user looks at).

5.1 Predicting gaze with head fixation

We make a test of eye gaze tracking with the user's head stable. The user holds his head still when makes the procedure of calibration and then performs the eye gaze tracking while still keeping his head stable in that position. We make the test of eye gaze tracking with 16 others points, the result is shown in Figure 4, the points red are points target of test, the points blue are points predictions of eye gaze. We can see in the figure 4, the results are excellent. But,

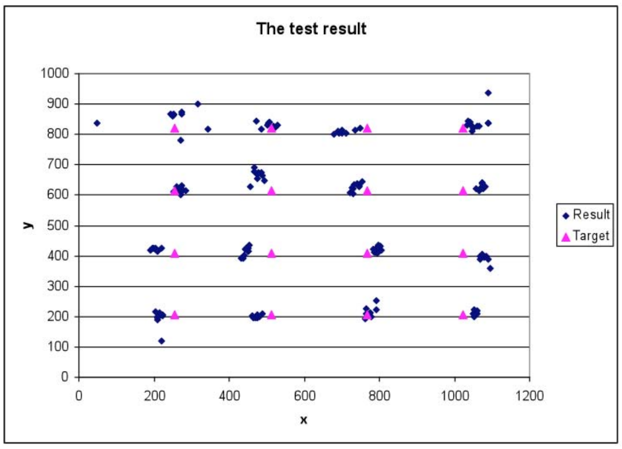

Figure 4: The result of test with head stable.

this result is good only when user holds his head stable. If user moves his head away from original position where user performs the eye gaze calibration, the accuracy of these eye gaze tracking systems drops dramatically. The reason is that when user moves his head, the image of the user's eye has changed. And the image of the eye input will be different to the image of the eye in training set. It makes the result of the prediction of the eye gaze goes wrong. We propose a solution for this problem in the next section.

5.2 Predicting gaze with head movement

To solve the problem of eye gaze tracking while moving user's head, we propose an idea of repeating the procedure of calibration in the differences angles view of the user. With this method, we add more data training to the training set. That means we add the various image of the eye in the different positions of user's head while looking the same point on the screen. After the procedure of calibration, user cans move his head freely while predicting the eye gaze.

We make a test with the same procedure of calibration in the previous section but repeat it four more times under the different angles of view. Two procedures of calibration perform with the head of user rotates to the left and to the right. Two others perform with the head of user rotate up and down. The degree maximum of the rotation of each direction is the position that all the corners do not lost of tracking. The figure 5 shows the four positions of user's head to make the calibration.

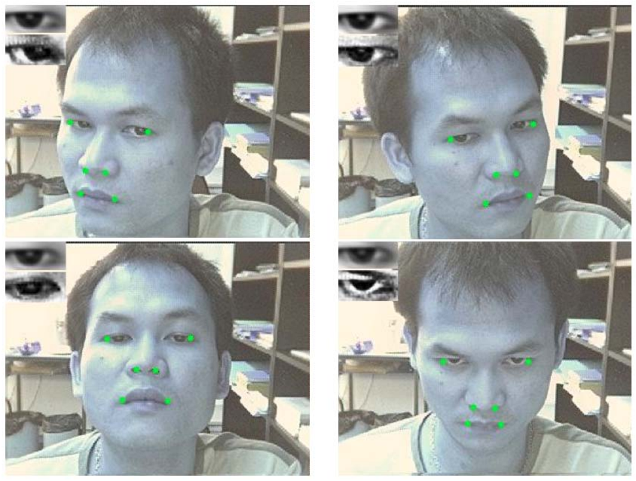

Figure 5: Four position of user's head to make four more times of calibration.

After calibration, we make the same test of predicting the eye gaze with 16 points like the test in previous section but under the condition of moving freely the user's head. User rotates his head from left to right or from down to up while looking at each point. The result is shown in the figure 6. We can see that the result in the figure 6 is not very good, the errors of eye gaze is large. The reason is that, there are still some positions of visual view of the user's head which make the image of the eye input different to the eye in training set. So we perform five more times calibrations at the positions that make the prediction of the eye gaze is not correct to add more data to the training set. And we do the test again. The result is shown in the figure 7. We can see that, after performing 10 times of calibration, the result of eye gaze predicting while the user moving his head is very good. We can even make this result better if we make more of calibrations.

5.3 Discussion

With 10 times of calibration in difference positions in the

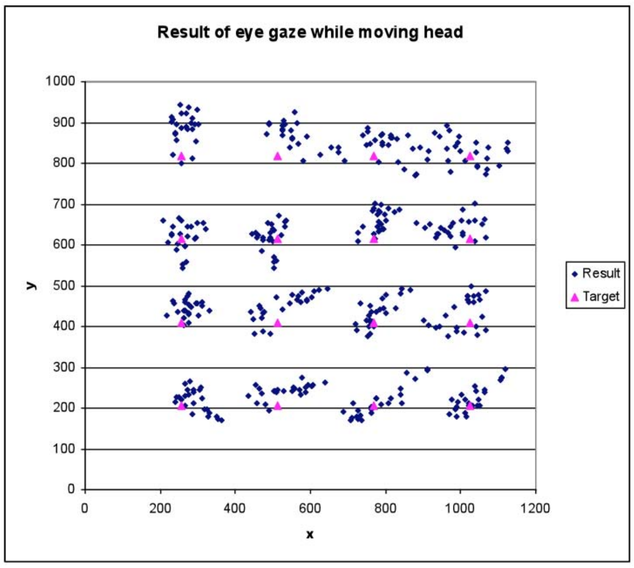

Figure 6: The result of test while moving head.

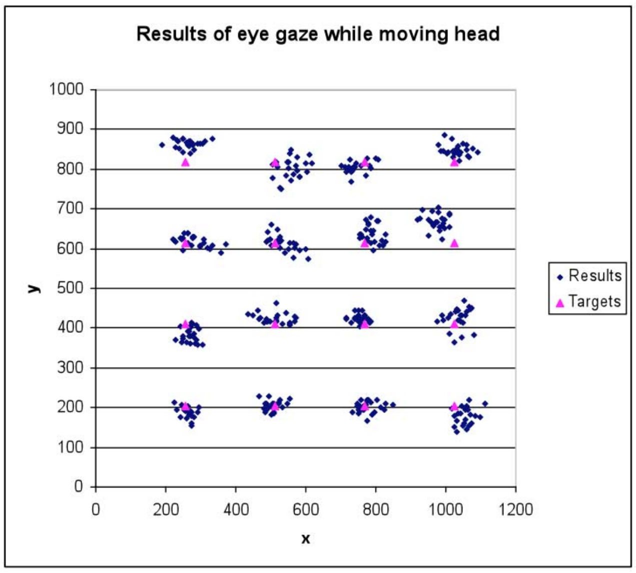

Figure 7: The result of test while moving head.

test above, the result is very good in the figure 7. But actually, the result will not be good if the user's head moves in to a new position (out of 10 positions that make calibration), for example user do not rotate his head but moves his head far away to camera or close to camera. In that case, we have to make more the calibration with the new positions of the user's head. So in the time of tracking the eye gaze under the natural movement of the head, if there is a position that makes the prediction of the eye gaze is not good, user cans make more the procedure of calibration at that position to make the result of the system of eye gaze tracking more accuracy.

Perhaps we can think that with this drawback, this system of eye gaze tracking become difficult and complicated to use for the users. But actually in practice, in the first time of using this system, user takes about tens minutes for calibration in all of position where his head wants to move to while making the prediction of the eye gaze, and the user can save this data of calibration. From the next time of using he cans reuse it and does not have to do any more the calibration. So it is very useful and easy for using in practice.

6. CONCLUSION AND FURTHER RESEARCH

We have presented a non-intrusive approach for eye gaze tracker in real time under head natural movements with a simple camera. Firstly, we use Haar-like features to detect the eye of a person in camera. Then we use algorithm Lucas Kanade to track the eye in real time and use the Outliers detection method to detect the errors of tracking and recover them. Lastly, we have used the Gaussian process to make predictions of the eye gaze in real time. Our system has following advantages:

- To use this system of eye gaze tracking a user needs only one camera simple or the camera built-into almost of computer portable nowadays.

- To predict the eye gaze a user spends only about ten minutes for the procedure of calibration with difference positions of user's head for the first time of using. That is his own database of calibration to make the prediction of his eye gaze. From the second time of using, user cans reuse it and he does not need to perform the procedure of calibration again.

- To predict the eye gaze a user spends only about ten minutes for the procedure of calibration with difference positions of user's head for the first time of using. That is his own database of calibration to make the prediction of his eye gaze. From the second time of using, user cans reuse it and he does not need to perform the procedure of calibration again.

7. REFERENCES

[1] M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25(6):821–837, 1964.

[2] D. Beymer, M. Flickner, I. Center, and C. San Jose. Eye gaze tracking using an active stereo head. 2, 2003.

[3] J. Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm. *Intel Corporation, Microprocessor Research Labs, OpenCV Documents*, 3, 1999.

[4] C. Harris and M. Stephens. A combined corner and edge detector. 15:50, 1988.

[5] D. Hawkins. *Identification of outliers.* Chapman & Hall, 1980.

[6] C. Hennessey, B. Noureddin, and P. Lawrence. A single camera eye-gaze tracking system with free head motion. pages 87–94, 2006.

[7] R. Lienhart and J. Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. 2(1):900–903, 2002.

[8] C. Morimoto, D. Koons, A. Amir, and M. Flickner. Frame-rate pupil detector and gaze tracker. 99, 1999.

[9] M. K. S. Nabil M. Hewahi. Class outliers mining: Distance-based approach. *International Journal of Intelligent Systems and Technologies 2*, Winter 2007.

[10] B. L. Nguyen, Y. Chahir, and F. Jouen. Free eye gaze tracking using gaussian processes. *to be appeared in conference proceedings IPCV*, 2009.

[11] T. Ohno and N. Mukawa. A free-head, simple calibration, gaze tracking system that enables gaze-based interaction. pages 115–122, 2004.

[12] T. Ohno, N. Mukawa, and A. Yoshikawa. FreeGaze: a gaze tracking system for everyday gaze interaction. pages 125–132, 2002.

[13] A. Perez, M. Cordoba, A. Garcia, R. Mendez, M. Munoz, J. Pedraza, and F. Sanchez. A precise eye-gaze detection and tracking system. 2003.

[14] C. Rasmussen, C. Williams, and I. Books24x7. *Gaussian processes for machine learning.* Springer, 2006.

[15] P. Viola and M. Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. 1, 2001.

[16] D. Yoo and M. Chung. A novel non-intrusive eye gaze estimation using cross-ratio under large head motion. *Computer Vision and Image Understanding*, 98(1):25–51, 2005.

[17] Z. Zhu and Q. Ji. Eye gaze tracking under natural head movements. 1, 2005.