GazePointer: A Real Time Mouse Pointer Control Implementation Based on Eye Gaze Tracking

Muhammad Usman Ghani, Sarah Chaudhry, Maryam Sohail, M. Nafees Geelani
COMSATS Institute of Information Technology
Lahore, Pakistan
{ch.usman.ghani, ngeelani48}@gmail.com, comsian_s9@hotmail.com, maryamsohail@live.com

ABSTRACT: *The field of Human-Computer Interaction (HCI) has witnessed a tremendous growth in the past decade. The advent of tablet PCs and cell phones allowing touch-based control has been hailed warmly. The researchers in this field have also explored the potential of 'eye-gaze' as a possible means of interaction. Some commercial solutions have already been launched, but they are as yet expensive and offer limited usability. This paper strives to present a low cost real time system for eye-gaze based human-computer interaction.*

Keywords: Eye Gaze Tracking, Human Computer Interaction, Computer Vision, Low Cost, Real-time

Received: 19 February 2014, Revised 27 March 2014, Accepted 2 April 2014

1. Introduction

Innovative and efficient techniques of HCI are being developed rapidly. It is an active research field of many experts. This paper concentrates on a human computer interaction application based on eye-gaze tracking. Human eyes carry much information which can be extracted and can be used in many applications i.e. Computer Interaction. Eye gaze reflects a person's point of interest. Eye gaze tracking is aimed to keep track of human eye-gaze. "*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*" [1]. This can be achieved by employing computer vision and image processing algorithms.

Technique explained in the paper is non-invasive and userfriendly, as it does not require a complex hardware or wires. Moreover, it does not have any physical interaction with the user. A cheap solution is provided for gaze-tracking. A built-in web-cam in laptop is used as a capturing device. A software based solution is proposed for controlling mouse pointer using '*eye gaze*'.

It is a natural and efficient way of interaction with the computer. Mostly the methods of interaction available are complex and cumbersome. Using this method, for controlling mouse pointer increases the interaction efficiency and reduces complexity. An illustration of setup for GazePointer is presented in Figure 1.

This technique is a special boon for disabled persons, such as spinal cord injured, or paralyzed patients. These patients are entirely dependent on assistance. Currently, disabled people usually type on the computer keyboard with long sticks that they hold in their mouth [2], but the technique being presented is a benefaction for handicaps to help them be independent in their lives. Giving them a chance to work, socialize, and entertain in their lives.

A number of eye-gaze tracking techniques are already available. Some of the techniques are Electra-Oculography [3] Laser based tracking [4], Corneal Reflections [2], Purkinje Image Tracking [5] and Head Mounted Techniques [6].

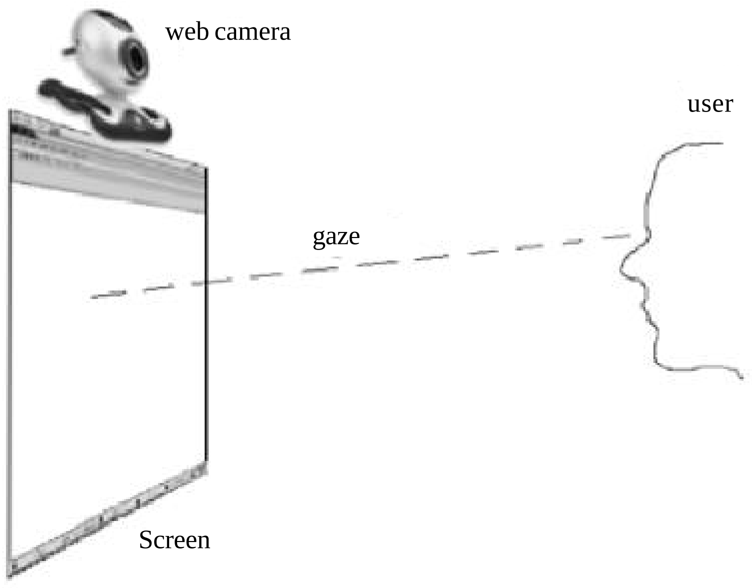

Figure 1. An illustration of Setup for Eye Gaze Tracking [7]

2. Eye Gaze Tracking Algorithm

Eye gaze tracking algorithm overview is presented in Figure 2. It consists of three major modules: (i) Facial features extraction; (ii) Eyes features detection and (iii) Point of Gaze calculation.

Algorithm presented in this paper performs operations on grayscale images. Camera captures BGR color space images, which is default color space for OpenCV. As a first step BGR → grayscale color space conversion is performed. Basic image pre-processing procedures are performed at each stage of algorithm. Histogram equalization is applied on grayscale images to normalize contrast in acquired image. It attempts to equalize the image histogram by adjusting pixel intensities in accordance with histogram [8]. For face detection, a machine leaning based approach is used, Object detection algorithm proposed in [9] is implemented in OpenCV library. This technique employs a Haar-features based approach for object detection, which makes the rapid and accurate object detection possible. Eye patch extraction can also be performed using above object detection algorithm described in [9] and its implementation is available in OpenCV library. Before extracting eye features, image pre-processing operations are carried out on each frame. For pupil detection, extracted eye patch must be smoothed to avoid false detections. It is necessary because pupil detection technique being used is based on Hough Circle Transform [10] which takes binary image as input. For image binarization, edge detection approach is used. Eye region being used to trace the '*Test Area*' is to be detected, for this purpose a simple calibration technique is designed, which is explained in section III. After features detection, a simple Point of Gaze calculation algorithm is designed which systematically interrelates the detected feature points to result in a precise PoG calculation.

2.1 Facial features extraction

Face detection algorithms are being employed in a wide variety of applications. This paper intends to present an eye gaze tracking algorithm and facial features detection i.e. face & eyes extraction is an important task in this regard, as it is the first module of features detection pipeline, errors at this stage may amplify and cause major drift in PoG calculation.

2.1.1 Histogram Equalization

Histogram equalization is a technique for contrast normalization by adjusting image intensities in accordance with image

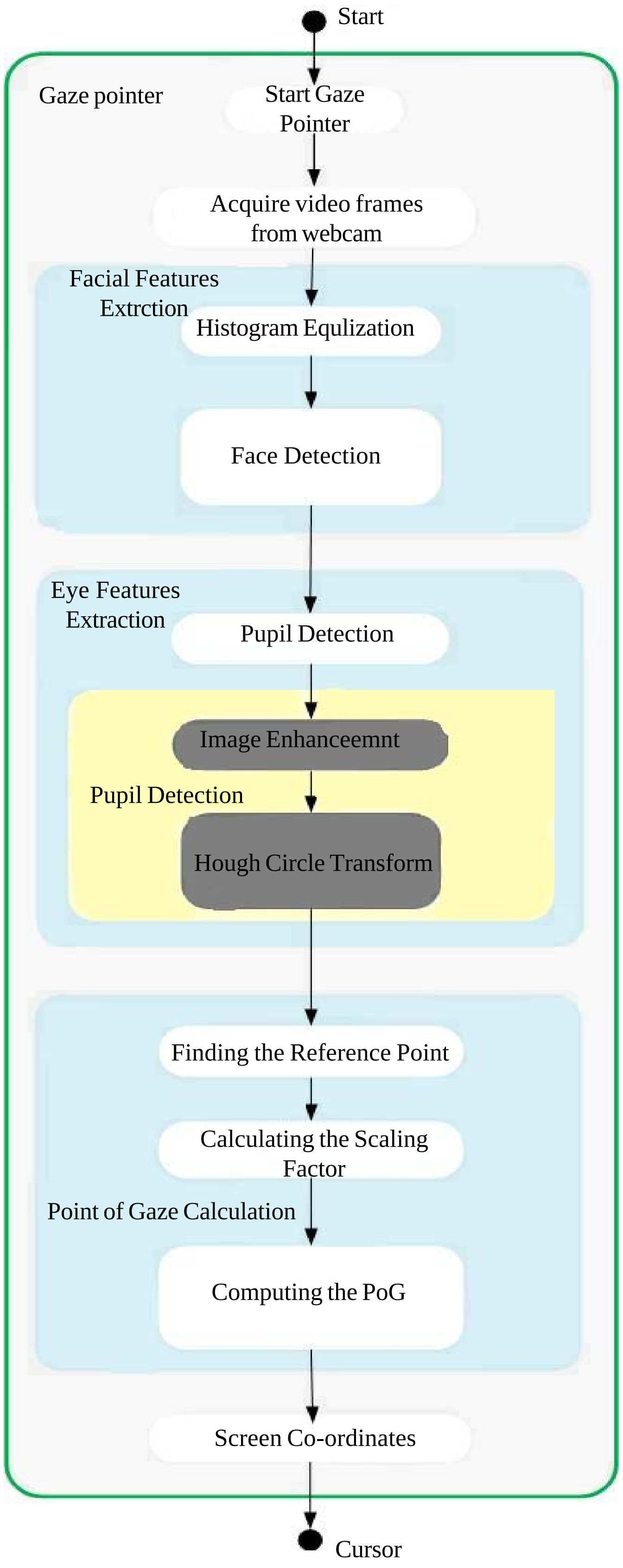

Figure 2. Overview of GazePointer algorithm

histogram [8]. It is a contrast enhancement procedure which takes benefit of histogram data and attempts to equalize the histogram. For performing facial features detection, histogram equalization gives benefit of better performance in terms of accuracy. As Face detection implementation available in OpenCV library requires the image histogram to be enhanced.

2.1.2 Face and eye patch extraction

Face detection is a classification problem i.e. classify acquired image into face and non-face regions. A rapid and robust object detection algorithm is employed to perform face and eyes extraction, proposed in [9]. OpenCV library provides an implementation of this algorithm; it follows a machine learning based approach. Haar-features based classifiers are trained for several objects detection i.e. face, eyes, nose, and human body. These classifiers contain training data for false positive and false negative samples. A set of simple features are obtained from training data. Haar- features are calculated by the difference between dark-region and light-region pixel values. A threshold value is fixed at learning stage i.e. feature is said to be present if difference value comes out to be greater than the value fixed as threshold.

Face and eyes detection is a complex procedure, require much computations to solve this classification problem. For this purpose, acquired images are down-sampled, face is detected and face co-ordinates are mapped to original image using simple calculations. Same practice is exercised for eye patch extraction to reduce the computation time; this approach has proved to be effective in order to achieve real-time processing of the frame.

2.2 Eye features detection

Eye gaze tracking is a complex problem; it needs to acquire a number of facial and eyes features to compute Point of Gaze (PoG). In this regard, first problem is to identify necessary and sufficient eyes features which can result in an accurate PoG calculation. While features identification, it should also be kept in mind that number of feature points should be as much less to result in a smooth computer interaction. This paper presents a simple and real-time eye gaze tracking algorithm. Two important eye features necessary to compute PoG were identified, which are (i) Pupil and (ii) Eye Corners. This section presents the techniques utilized for these eye features extraction.

2.2.1 Pupil Detection

Pupil is the central and focusing part of eye, located at center of iris. Light enters into eye through pupil, and finding the position of pupil is principal point of interest in proposed technique. Eye gaze projection is based upon the relative displacement of pupil from center of eye. Pupil needs to be detected to project user's eye gaze in the '*Test Area*'. The first step in this regard, is to detect the iris from the frames captured with webcam, and then pupil can be found, as it is situated at center of iris.

Iris is the flat, colored and circular part on the white region (Sclera) of an eye. Iris varies in color as black, blue, green etc. Iris is a circular region and can be detected using the very commonly used Hough circle transform technique [8].

Hough Circle Transform takes a binary image as an input and detects circle in it. The quality of the image needs to be good to extract every possible information from it. First, the input image is enhanced for good quality and then Hough Circular transform" is applied on it.

i. Image Enhancement

To enhance image, smoothening is applied which in-effect reduce noise. For this purpose, grayscale image is passed through a blurring filter, gaussianBlur [11] is applied available in OpenCV library.

ii. Hough Circle Transform

The technique used for iris detection is '*Hough Circle Transform*'. This paper uses OpenCV implementation of Hough Circle Transform [12]. HCT routine takes grayscale image as input and applies '*Canny*' edge detection to compute a binary image. The reader is assumed to study HCT documentation available at [12], function parameters must be computed with care, as a slight change in parameters may increase or decrease accuracy of algorithm depending upon the scenario or application.

2.2.2 Eye Corners Extraction

Corners extraction has always been a point of interest in computer vision. It has been interesting as they are two dimensional features and could be easily detected. In this paper eye corners are being detected, as they are an important feature and are necessary to calculate Point of Gaze. This is performed on the contours of the eye therefore it becomes difficult to find the position of eye corners repeatedly, as they are to be found on the same image.

Eye corners are such an important eye feature that a lot of information about eyes can be extracted using eye corner locations. Once eye corner locations are known, they can be used to calculate eye width, eye height and most importantly this information can be used to locate center of eye.

Another fact is established through several experiments that eye corners extraction is not suitable, to get to know about movable area for iris in eye to trace the '*Test Area*'. For this purpose, after analysis it was concluded that computer is not that intelligent to discover those boundary points in eye, user should feed some information at start and some procedures must be implemented to use this information in intelligent way to extract features i.e. movable area for iris, center of eye etc. To meet this requirement, a simple 4-point calibration technique is designed which is explained in next section.

i. Calibration

Calibration techniques are designed to aid the systems in correctly calculating the PoG. Decision about number of calibration points is critical. An effective calibration algorithm must have as much number of calibration points which can make the user familiar with system and on the other hand it should also be simple enough that it would not cause nuisance to the user.

A simple 4-point calibration algorithm is designed for this system. Idea behind designing such calibration algorithm is, it can be helpful to calculate eye region which will be used to scan the '*Test Area*'. This simple algorithm allows the user to look at all corner points in free mode. Here '*free mode*' suggests that user is allowed to stay at a corner point for an arbitrary time duration. This idea helps the system to reduce errors occurred due to erroneous pupil detections during calibration.

It starts by letting the user look at top left corner of '*Test Area*', meanwhile system starts its detection cycle, starting from face detection to pupil detection, system stores detected pupil locations until user clicks again in the GUI, system stops saving pupil location results, after averaging the stored pupil locations, system stores the averaged results as top left corner location of eye region scanning the '*Test Area*'. This process continues for rest of three corner points and as soon as calibration ends, it enters in main Eye Gaze Tracking loop which let the user interact using eyes.

2.3 Point of Gaze Calculation Algorithm

Point of gaze can be referred to as the point of interest of user in '*Test Area*' he/she looking at or gazing at. User's point of interest i.e. PoG can be calculated by extracting eye patch and some important eye features. Important eye features sufficient to calculate PoG has been identified and discussed in earlier sections.

It is the most important and critical stage of algorithm as it involves using already found feature points to calculate PoG. This stage must effort to compensate the errors occurred at detection stage.

2.3.1 Finding the Reference Point

It is absolutely infeasible to perform PoG calculations without a reference point. It can be helpful in PoG calculations because less calculations will be required to translate pupil movements in eyes into cursor movements on screen. '*Centre of Eye*' can act as a desired candidate for reference point. Eyes movable region has already been computed during calibration stage, a simple averaging of *x* and *y* co-ordinates can result in Centre of Eye calculation. This concept is illustrated in Equation 1 and Equation 2.

$$COE_x = (TopRightCorner_x + TopLeftCorner_x) / 2 \qquad (1)$$

$$COE_y = (TopRightCorner_y + TopLeftCorner_y) / 2 \qquad (2)$$

Where, *COEx* and *COEy* denote *x* and *y* coordinates of center point of eye's movable region respectively. *TopRightCorner*, *TopLeftCorner*, *BottomRightCorner* and *BottomLeftCorner* construct a rectangular region which represent eye's movable region.

2.3.2 Calculating the Scaling Factor

In this step the cursor movements and pupil movements were interrelated i.e. it was to be found that how many pixels a cursor will traverse for a single pixel movement of the pupil. For this calculation width and height of eyes were associated with the width and height of the screen. Screen width and height is constant, but eye's movable region width and height is subject to change in different scenarios. Eye's movable region width and height can be computed using Equation 3 and Equation 4.

$$W_{eye} = TopLeftCorner_y – TopRightCorner_y \qquad (3)$$

$$h_{eye} = TopRightCorner_y - BottomRightCorner_y \quad (4)$$

Where, w_{eye} and h_{eye} represent width and height of eye's movable region respectively. Now scaling factor is to be computed for x and y coordinates with help of Equation 5 and Equation 6.

$$R_x = W_{screen} / W_{eye} \quad (5)$$

$$R_y = h_{screen} / h_{eye} \quad (6)$$

Where, w_{screen} and h_{screen} denote width and height of '*Test Area*'. *Rx* and *Ry* represent scaling factor for x and y coordinates.

2.3.3 Computing the PoG

This is the final step of POG calculation as well as GazePointer algorithm. This stage will realize the significance of reference point. It translates the pupil movements in eyes into cursor movements in '*Test Area*'. Taking assumption that reference point in eye corresponds to center point in '*Test Area*', pupil movements can be simply translated into cursor movements using Equation 7 and Equation 8.

$$PoG_x = \frac{W_{screen}}{2} + R_x \times r_x \quad (7)$$

$$PoG_y = \frac{h_{screen}}{2} + R_y \times r_y \quad (8)$$

Where, PoG_x and PoG_y represent x and y coordinates of Point of Gaze respectively and r_x denotes pupil distance in x direction from reference point and r_y denotes pupil distance in y direction from reference point and they can be computed by using Equation 9 and Equation 10.

$$r_x = COI_x - COE_x \quad (9)$$

$$r_y = COI_y - COE_y \quad (10)$$

Where, *COI* represents pupil location. Figure 4 illustrates this procedure.

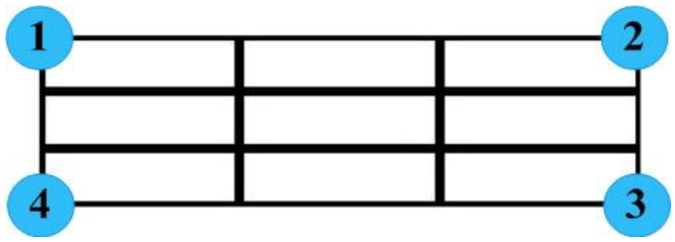

Figure 3. 4-Point Simple Calibration

2.3.4 Limiting, Smoothing and Controlling Speed of Cursor Movements

Cursor should always remain inside the '*Test Area*', this fact must be incorporated into GazePointer algorithm. This implies that a condition should be applied on cursor position which limits the cursor inside the '*Test Area*'. This can compensate the erroneous detections at features detection stage. Another important concept is introduced in GazePointer, by applying a threshold to cursor movements. This is achieved by experimentally finding a threshold such that user doesn't move his/her eyes more than threshold in a frame processing time. False projections can be reduced which can improve GazePointer accuracy.

Hardware mouse provides smooth user interaction, GazePointer should also offer smooth interaction. For this purpose, a simple concept is introduced that cursor should not directly jump to next location but it must be dragged from previous location to next location.

This can be achieved by applying an iterative procedure, using this procedure user can be allowed to control speed of

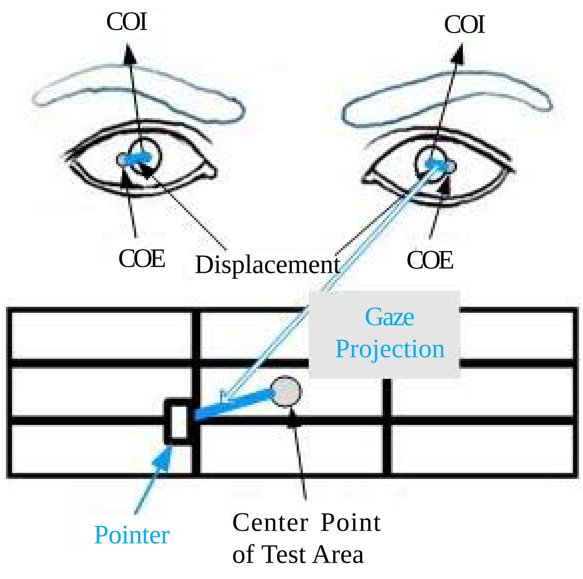

Figure 4. Computing Point of Gaze using Reference Point

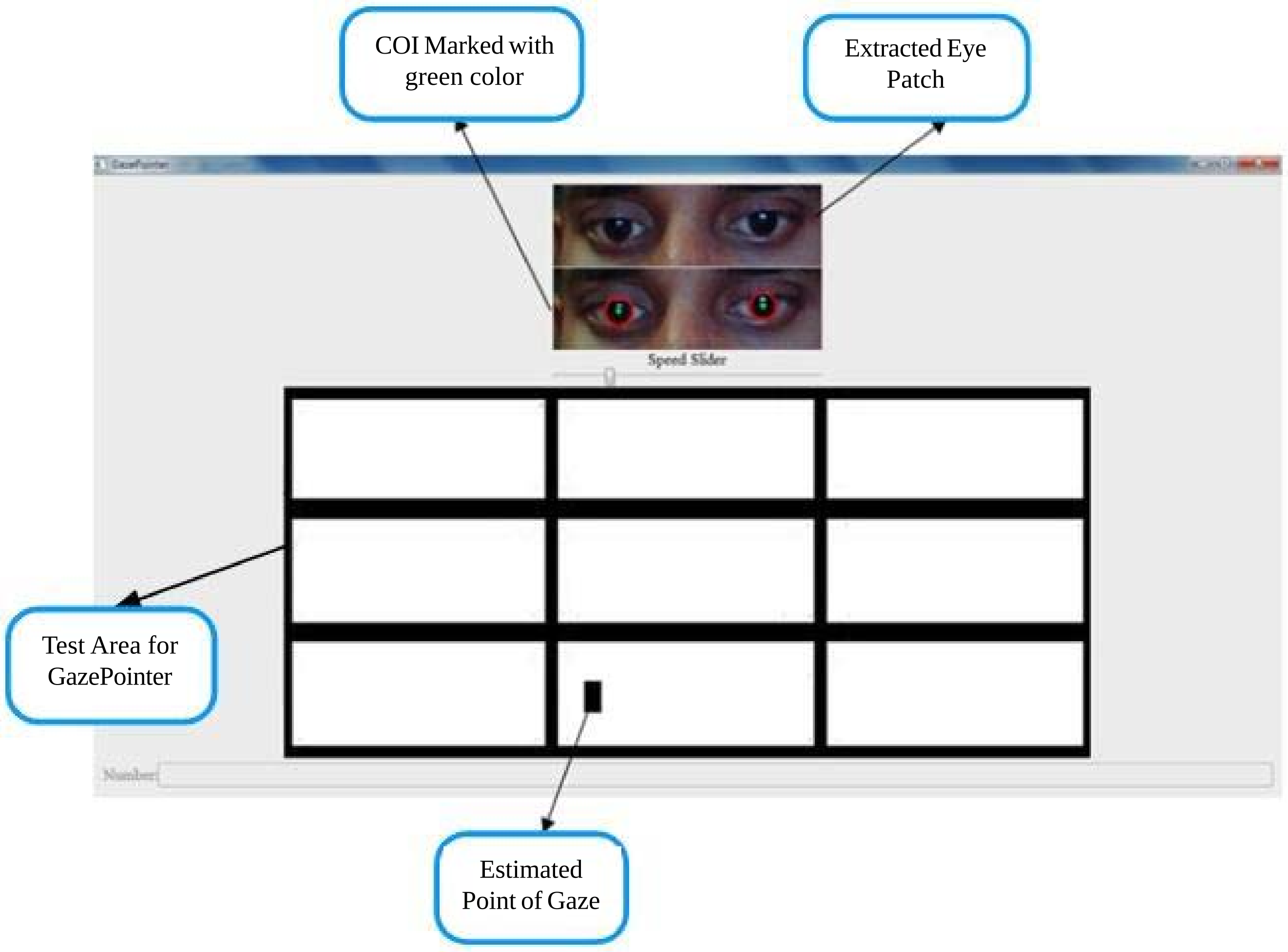

Figure 5. GazePointer GUI

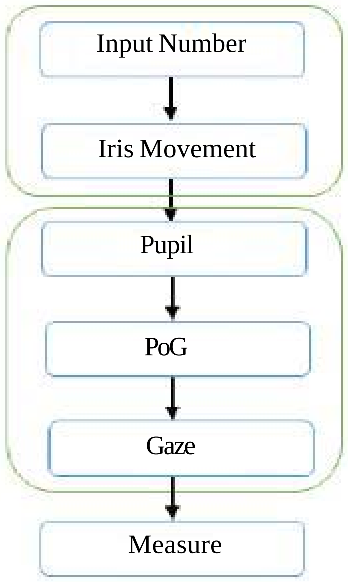

Figure 6. Algorithm overview of Test Bench

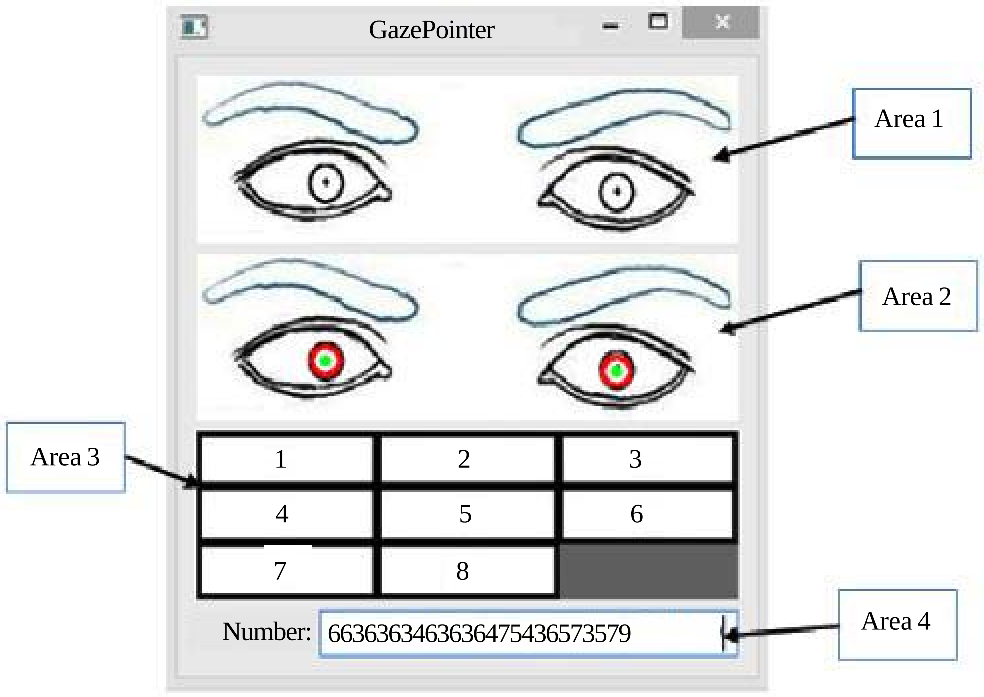

Figure 7. Test Bench GUI

the Mouse Pointer using a speed control knob. These add-ons help GazePointer to offer user friendly Human-Computer Interaction.

3. GUI Designing

A '*Graphical User Interface*' is designed to demonstrate the results. Qt framework [13] is utilized for designing GUI. It has three basic areas.

First, two areas show the video extracted from the webcam and other indicates HCT results. The portion under it represents the area for movement of mouse pointer. A small GazePointer is shown in the figure which shows the projection of eye movements.

4. Test Bench

Quantifying the accuracy of an algorithm or system is always necessary to justify its usability in real world applications. Test bench models help to quantify the accuracy of a system/algorithm in a simulated or ideal environment. Test bench algorithm flow is presented in Figure 6.

A simple Test Bench model was designed to test the accuracy of this system. Artificial eyes instead of human eyes were used. Application prompts user to input a number. Iris moves in accordance with the number entered. Algorithm processes these iris movements and projects the eye gaze in Test Area. An image of this Test Bench GUI is given in Figure 7. Description of different area of Test Bench GUI is given below:

- Area1: Artificial eyes
- Area2: Processing results
- Area3: Test Area for Gaze Projections
- Area4: Text Box displaying region number

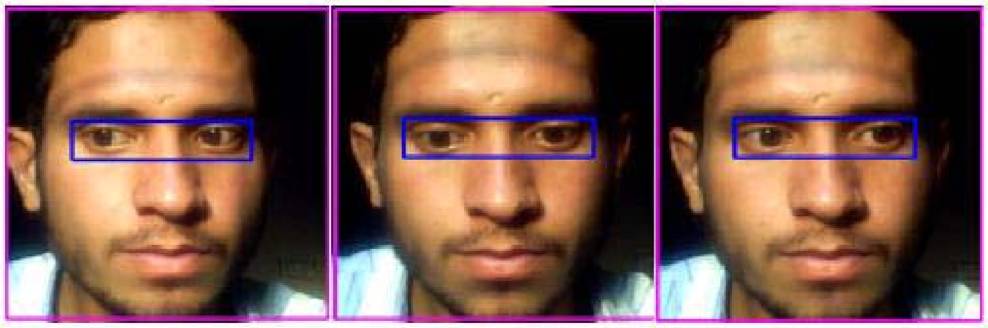

Figure 8. Face and Eyes Detection Results in different frames

5. Results and Evaluation

In this section, results of GazePointer implementation are presented. Several experiments were performed to achieve these results.

5.1 Face and Eyes Detection

Face detection provided satisfactory results. Accuracy does not drop with changing lighting conditions, backgrounds, and distance. This implementation detects frontal faces only. Its accuracy remained 100% when tested on a limited dataset of 223 frames. Face detection results are presented in Figure 8.

GazePointer is assumed to be a system working with realtime constraints. Captured frames were down sampled 5 times to reduce the computation time. Before resizing, it took 500-600 mS per frame and it reduced to only 13-15 mS per frame.

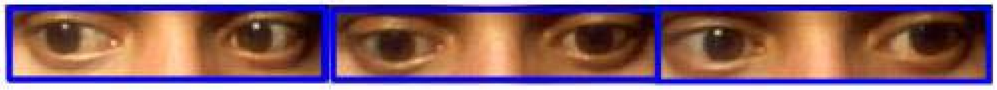

Figure 9. Extracted Eye Patch in different frames

Eyes detection implementation was done accurately in almost every case when face was detected. This implementation as also tested on a limited dataset of 223 frames and it resulted in 100% accuracy. Eyes detection results are given in Figure 9.

This implementation resulted in a large computation time. The computation time was reduced by limiting the region of interest to detected face only. Then this face region was down sampled 2 times. Computation time was reduced to 10-15 mS per frame after modifying the implementation.

5.2 Pupil Detection

HCT implementation was quite light weight and resulted in a few mS processing time per second. Its initial algorithm resulted in lots of false detections. It took 6-10 mS per frame for processing and resulted in accuracy of 80% when tested on a limited dataset of 223 frames. Pupil detection results are given in Figure 10.

The rate of false detections was decreased by applying a threshold between previous frame pupil location and present frame pupil location. Thus, accuracy improved to 87%.

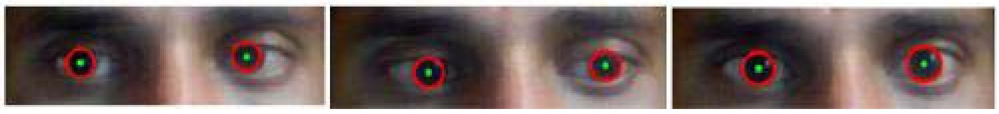
Figure 10. Pupil detection in different frames

5.3 Point of Gaze Calculation

Point of Gaze Calculation resulted in satisfactory results when tested on a limited dataset. Complicated results were acquired while testing on live stream frames acquired from webcam. PoG calculation results are presented in Figure 11. Projections followed user's eye-gaze often. False detections were involved because pupil detection results were not 100%. Pointer size was quite large due to low resolution webcam of laptop.

System Limitations

Proposed system performance was analyzed in different scenarios and some limitations were defined, which are given below:

• User head should be at the same altitude as the webcam.

• Distance between user's eyes and webcam should be in the range of 20-75 cm.

• This system can't be used with glasses on.

• Lighting conditions should be extremely good.

6. OpenCV for Eye Gaze Tracking

Eye gaze tracking for Human computer interaction requires real-time processing of images, being acquired through webcam. It necessitates the system to be extraordinarily computationally efficient and algorithm must work at least at a frame rate of 15 fps. Real-time processing of frames is necessary to provide user-friendly and smooth interaction. It becomes a challenging task when each image has to pass through a pipeline of complex operations. Eye gaze tracking for interaction is one of such problems; user's POG is changing at every moment, which must be tracked in realtime.

OpenCV [14] is a library of image processing algorithms by Intel®. It was designed for computational efficiency, with a strong focus on real-time computer vision applications. It provides a wide range of algorithm implementations in its C, C++, Java & Python interfaces and it supports MS Windows, Linux, Mac OS, iOS and Android [14].

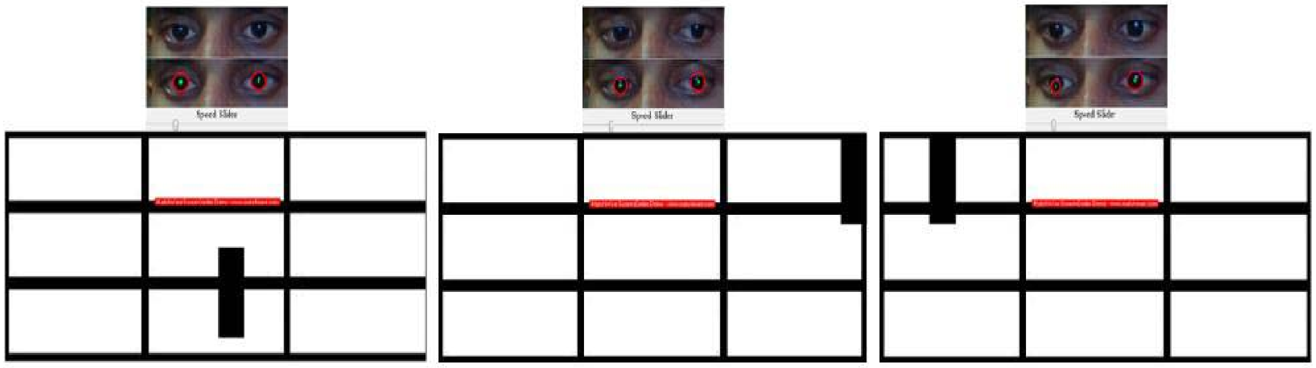

Figure 11. PoG calculation results in different frames

7. Conclusion

Eye gaze tracking is a complex problem and there can be many solutions to address this problem. In this project a computer vision algorithms based solution is implemented. A low cost, real-time solution for eye gaze tracking is developed with help of OpenCV library. There are many applications of eye gaze tracking, for instance in HCI, appliances control, usability studies and in advertising effectiveness.

Accuracy for features extraction algorithms depend on image quality and lighting conditions. Algorithm performance drops down in poor lighting environment. Computer Vision algorithms are employed for features detection and they don't perform well in lighting environments discussed earlier.

PoG is accurately calculated provided detections are correct, only a few erroneous PoG projections were found and reason was false detections in features detection stage. Pointer size is large due to low resolution and small Test Area size.

8. Future Work

To improve the projection results, image quality must be enhanced. Better image quality would improve accuracy of computer vision algorithms. Sophisticated pre-processing algorithms should be introduced to compensate lighting variations and webcam resolution should also be increased to decrease the pointer size.

A feature describing head-posture must also be introduced, it will allow the user to move-freely while interacting with system.

Introducing the concept of gaze estimation along with gaze projection will be beneficial because it will improve gaze projections drastically. This idea is of more importance because it will cause a tremendous improvement in user experience, as the idea of gaze estimation promises to learn from usage statistics and infer gaze projections. Particle Filters should be preferred to implement gaze estimation because they are quite simple and has resemblance with problem of gaze estimation.

GazePointer can be extended to many applications i.e. eyegaze based interfaces, appliance control. It can also be extended to different Operating Systems like MS Windows®, Linux and Mac OS.

9. Acknowledgment

Authors would like to acknowledge the support and services of IEEE Research and Development Program Pakistan (IEEE RDPP) for creating awareness about research and development around the region.

References

[1] Poole, Alex., Ball, Linden J. (2006). Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future Prospects, in Encyclopedia of Human Computer Interaction (30 December 2005), p. 211-219.

[2] Yoo, D. H., Kim, J. H., Lee, B. R., Chung, M. J. (2002). Noncontact Eye Gaze Tracking System by Mapping of Corneal Reflections, *In*: Fifth IEEE International Conference on Automatic Face and Gesture Recognition (FGRí02), p. 94-99.

[3] Singh, H., Singh. (2012). A Review on Electrooculography, *International Journal of Advanced Engineering Technology*, 3 (4).

[4] Irie, K., Wilson, B. A., Jones, R. D. (2002). A laser-based eye-tracking system, *Behavior Research Methods, Instruments, & Computers*, 34 (4) 561-572.

[5] Arai, K., Yamaura, M. (2010). Computer Input with Human Eyes-Only Using Two Purkinje Images Which Works in a Real-Time Basis without Calibration, *CSC Journals*, 1 (3) 71-82, 2010.

[6] Hua, H., Krishnaswamy, P., Rolland, J. P. (2006). Video based eyetracking methods and algorithms in headmounted displays, *Optics Express*, 1 (10) 4328-4350.

[7] Back, D. (2005). Neural Network Gaze Tracking using Web Camera., Linköping University, MS Thesis.

[8] Gonzalez, R., Woods, *R.* (2009). Digital Image Processing, Pearson Education.

[9] Viola, P., Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features, *In*: Computer Vision And Pattern Recognition.

[10] Kimme, C., Ballard, D., Sklansky, J. (1975). Finding circles by an array of accumulators, in Communications of the Association for Computing Machinery, p. 120–122.

[11] Image Filtering. (2002). [Online]. http://docs.opencv.org/modules/imgproc/doc/filtering.ht ml?highlight= gaussianblur#gaussianblur

[12] Hough Circle Transform. [Online]. http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html

[13] Qt Project. [Online]. http://qt-project.org/doc/qt-5.0/qtdoc/index.html

[14] Open Source Computer Vision Library (OpenCV). [Online]. http://opencv.org/