MASSEY UNIVERSITY

ALBANY NEW ZEALAND

Eye Detection and Accuracy of the Eye Mouse

Author: Jia Qi Zhao

Supervisor: Dr. Andre L. Barczak

Contents:

1. Introduction

Eye tracking is used to study users' attention patterns during task performance or to allow hands-free interaction with a computer for persons unable to use the traditional mouse and keyboard-based control inputs. As eye-tracking technology advances into the future, it can be clearly seen that it is advantageous to utilize eye-tracking as a replacement for traditional control tasks especially for disabled users.
In some cases, the use of eye tracking fits naturally with the intended task, for example a camera that utilizes the user's eyes to focus the lens at the location that the user is currently looking. At the same time, the performance of eye tracking equipment can vary due to many factors including low accuracy.

Tobii Dynavox [6], part of the Tobii Group, is a US-based company specialized in eye and touch tracking based technologies. They are an example of eye tracking advancements in the world, where they help develop systems for the disabled to use on a daily basis. The system is however costly averaging USD 1000 per eye tracking software. The products become more costly as their technology advances.

In this paper, results are presented to evaluate the typical accuracy of desktop eye tracking using a standard low definition camera. The basis surrounding this research project is to provide disabled users with this technology similar to Dynavox [6] eye tacking technologies as open source software improving accessibility and efficiency when using computer systems.

To meet this objective, I have developed a program using the C++ language and OpenCV[3] library that detects the user's face and pupils and displayed a visual representation on the screen. We have measured the program's accuracy, stability of detection.

This paper is organized as follows. First, an overview on relevant prior work, followed by the design of the experiment, then results are provided along with a discussion on the findings.

2. Literature Review

In 2001, Paul Viola and Michael Jones [2] proposed the first real-time object detection framework. This framework, being able to operate in real-time on 2001 hardware, was partially devoted to human face detection. And the result everyone knows - face detection is now a default feature for almost every digital camera and cell phone in the market. Even if those devices may not be using their method directly, this now ubiquitous availability of face detecting devices has certainly been influenced by their work. Their focus is primarily face detection which has been a problem in recent times.

Detecting images from a human point of view is easier than computer. A computer needs to accurately process pixels within images using instructions and constraints to identify the elements. The Viola-Jones [2] framework works in a similar manner being able to isolate features in an image to identify faces and non faces.

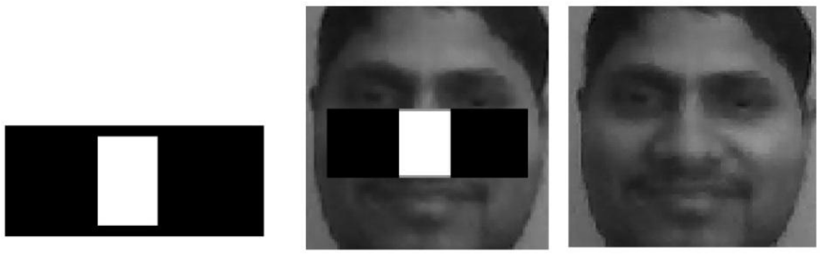
At any rate, the Viola-Jones [2] facial detection needs to be:

1. Robust which possesses high detection rate (true-positive)
2. Real-time (at least 2 frames per second)
3. Face detection (excluding recognition).

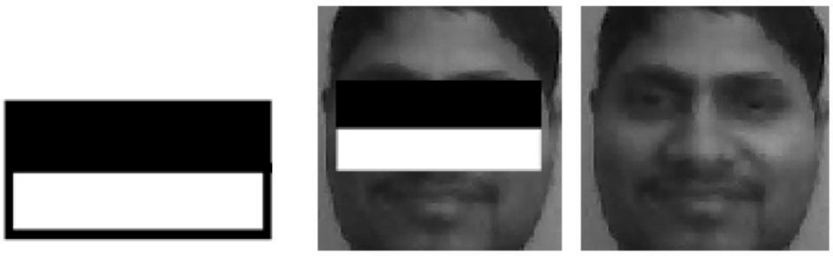
The Framework Algorithm consists of four (4) steps:

Step 1: Haar Features

All humans share similar facial properties and the Haar feature is able to match to these similarities when detecting faces. They are represented using rectangle features.



A Haar feature is applied on to specific location on the face where the nose bridge region is brighter than the eyes.



Another Haar feature is applied on to a specific location on the face where the eye region is darker than the upper cheeks. Modern day Haar features contain many different rectangles in all shapes and sizes to cover more features of a persons face.

Step 2: Integral Imaging

Evaluating rectangular features above is done by using the Integral Imaging which is able to detect area of a shaded(black) area of a rectangle feature by finding sum of pixels above and to the left of (x,y). This allows for calculation of sum of all pixels inside any given rectangle which can be used in Identifying Pupils (In Methodology).

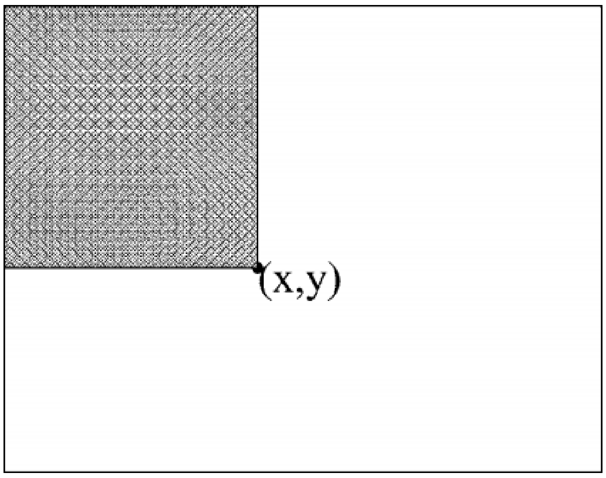


Figure 1 The value of the integral image at point (x, y) is the sum

Step 3: AdaBoost

It is a machine learning algorithm used to eliminate redundant features. Assuming all feature values will be calculated, testing and evaluating a standard 24 x 24 pixel sub window， approximately 160,000+ possibilities will be exhaustive and very expensive. Using AdaBoost training helps to classify and select best feature from all the possibilities to use them.

These best features are related to as weak classifiers [3] or a classification of features. This means they are able to clearly detect 50% of faces in a sample of faces supplied.

Adaboost will be able to establish a strong classifier [3] which is a combination of these weak classifiers hence increasing probability of detecting faces in a data population given (60-80% face detection)

Step 4: Cascading Classifiers

These strong classifiers [3] a grouped in stages and each stage contains a certain number of features to compare against test image. The job of each stage is to determine if a sub window is a face or not a face. If any of the stages fails, the sub window is disregarded as a face and it moves on. The further the increase in stages, the more complex the classifiers [3] become hence face detection accuracy increases.

The Viola-Jones [2] algorithm has a number of advantages, including extremely fast feature computation, efficient feature selection, scale and location invariant detection and the option to scale features instead of scaling the image.

The Viola-Jones [2] algorithm has some disadvantages. The algorithm only evaluates the frontal image and struggles to cope with rotation of the face in both vertical & horizontal axis and over 45degrees around the depth axis. It is also sensitive to lighting conditions and can, in error, detect the same face multiple times.

For this project, the Open CV library containing pre-trained classifiers [3] are used for face and pupil detections.

Basically the steps 1 – 4 discussed above are implemented in the OpenCV [3] library for uses however contain limitations and gaps. By combining the different pre-defined classifiers and making improvements on the algorithms, a simple eye tracking system may be created using a standard non-high definition web camera.

3. Materials

For this report, the following materials were used:
1. Developing/Testing Operating System: Windows 10 64-bit
2. Compiler Software: Open CV[3] and Code Block IDE [4]
3. Programming Language: C++
4. Logitech Web Camera (Logitech QuickCam Pro 9000)
5. Books: "*Open CV: Computer Vision with the OpenCV Library*" [1]
6. Stopwatch for timing
7. Open CV official Website documentation [3]

4. Methods

4.1 Haar Cascading Detection

To identify objects such as faces in images Haar feature-based trained classifiers such as haarcascade_frontalface_alt.xml and haarcascade_eye.xml are used due to the improved reliability and speed as opposed to other methods.
For both Frontal Face and Eye detection, the cascading OpenCV [3] methods are applied to detect facial feature locations. Each of the trained classifiers work in a cascade (gets more complicated further) manner and immediately fail if one of the stages does not pass. They contain rectangular features (Literature Review) present on the face to provide comparisons for example where dark rectangle to represent eye regions and lighter area for below the eye regions. These comparisons are done for each fps in real time.

4.2 Eye Location

Eye detection is based upon basic knowledge of eye location on a person's face. The OpenCV face detection algorithm is able to detect the face, hence required region of interest (ROI), this rectangle is reduced to only a small area, specifically the eye regions from 30% of the top part of the rectangle and 40% from the bottom part of the rectangle. This reduces processing time to identify the eye, increasing eye detection algorithm performance.

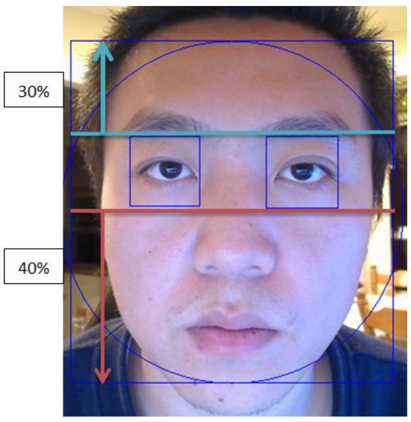
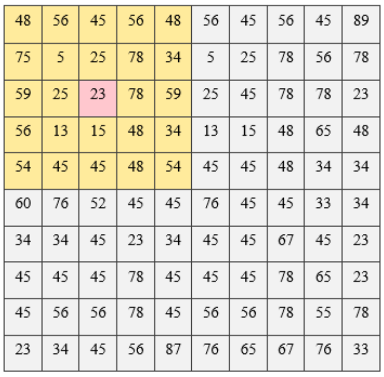


Fig: Top 30% and buttom 40% are not in the search area

4.3 Pupil Identification

Using the default face detection classifiers implemented within the OpenCV library, the face and eye outline of a users' face is detected. The rectangle containing the eye (from 4.2) is isolated and converted to grayscale for faster pupil identification. For each 5x5 pixel subsection of the rectangle, the average intensity is found. The lowest average found from all possible subsections is the (x,y) co-ordinates of the the darkest area of the rectangle, which identifies coincidentally the pupil.



48	56	45	56	48	56	45	56	45	89
75	5	25	78	34	5	25	78	56	78
59	25	23	78	59	25	45	78	78	23
56	13	15	48	34	13	15	48	65	48
54	45	45	48	54	45	45	48	34	34
60	76	52	45	45	76	45	45	33	34
34	34	45	23	34	45	45	67	45	23
45	45	45	78	45	45	45	78	65	23
45	56	56	78	45	56	56	78	55	78
23	34	45	56	87	76	65	67	76	33

4.4 Eye Mouse Algorithm

4.4.1 Algorithm 1: Eye Tracking via Range of Pupil Movement

Scaling eye movements to a standard 1280 x 960 monitor required establishing an array of pixels for the range of pupil movements. Users are required to follow instructions to initialize the range of movements to be detected by the computer. The range of movements was for each corners of the monitor (Top Right and Left, Bottom Right and Left). This would create a baseline or boundary to track the pupil. The range resembles a rectangle area of movement for the pupil and anything outside would cause inconsistencies.

Completing all required instructions initializes and scales all the values onto a black tracking window. This method allows monitoring small range of movements inside the boundary which are scaled onto tracking window multiplied by its ratio.

4.4.2 Algorithm 2: Eye Mouse Tracking via Face Location

The user has to initialize by centering the face onto the tracking window and ensuring eyes are also looking at the center of tracking screen.

The initial location is a half of the standard 1280 x 960 pixel screen display which is 640 * 480 pixels. The program initializes and scales by ratio, the face (A) coordinates onto the center of the tracking screen. The pupil is not initialized as we assume it will depend on face location. It is not scaled onto tracking screen, rather calculated based on the movement of the face location.

For tracking movements of pupil, new face location is calculated by getting the difference between new and old face location. The difference (Z) is scaled and added onto initialized face location to get new face location.

A = Scaled Initial Location (On tracking screen)
A.1 = Old Location
A.2 = New Location
S.R = Scale Ratio
N.L = New Location
Z = Difference

$$Z = A.2 - A.1$$
$$\mathbf{N.L = A + Z * S.R}$$

To attain new pupil values, locate and calculate for both pupils, the difference in (x,y) co-ordinates from the face(A) to each pupil and by scale, adding them to the difference(Z) which is new face location hence giving new pupil location according to new face location.

Example:
LP = Left Pupil
LP.1 = Difference
LP.NL = Left Pupil New Location
Difference in Pixel Coordinates
Difference (LP.1) = L.P (x, y) – New face (NL) (x, y)
LP.1 = ((LP. x – NL.x), (LP.y – NL.y)
Left Pupil New Location (LP.NL) = LP.1 x S.R
We also do for Right Pupil.

4.5 Testing Environment

4.5.1 Eye Mouse Accuracy Test

Testing the Eye Mouse algorithm involved recording the accuracy and error ratios of the algorithm on a test subject. A static test used a fixed distance between the subject's face and the camera was used to ensure a more accurate detection. When testing, the user was only required to move their head and eyes and the developer recorded the accuracy level of the tracking window. For each static test, the pixel pattern increased in size from a 2 by 2grid, through 3 by 3, 4 by 4, etc. until reaching an 8 by 8 pixel pattern.

4.5.2 Region Accuracy and Stability Test

Testing the region accuracy and stability involved initializing the test subject's gaze to the center of the screen. A 2 by 2 starting grid was used to start. Making sure the subject's head remained in static position, the subject pointed their head and eyes to specific regions on the tracking window inside the grid boundaries. The time it took for the on-screen pointer to move out of the boundaries was recorded, to a maximum time of 4 seconds. The program was then repeated with progressively larger gird sizes till a maximum grid size of 9 by 9 was reached.

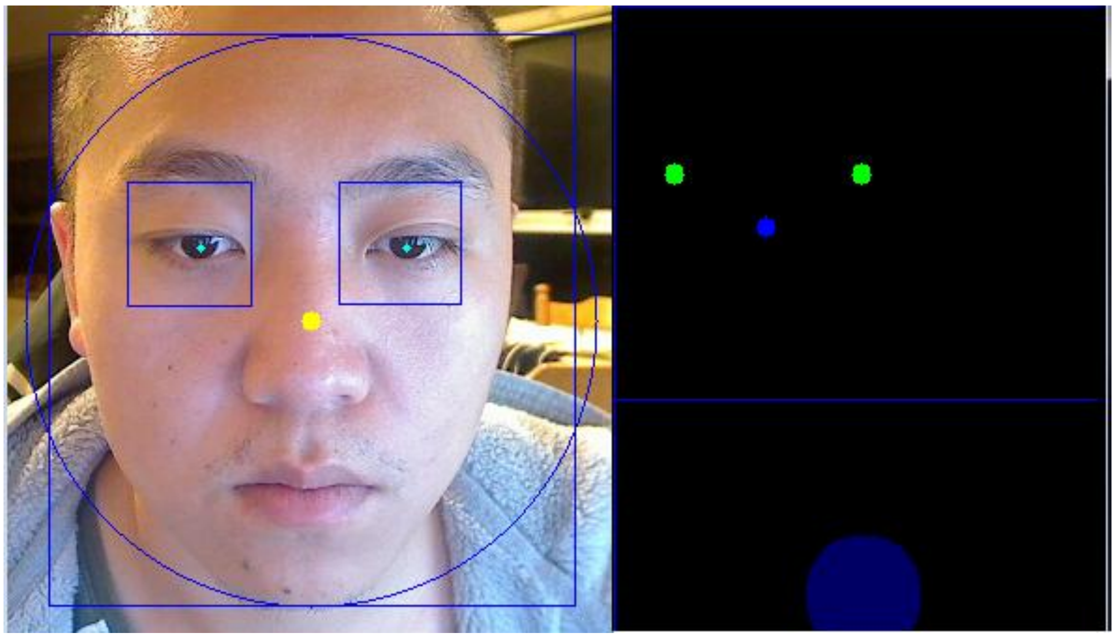


Fig: Method 4.2 Initializing face location from Camera generated image (Left) and Scaled onto the Tracking Window (Right). The Green dots from camera image are scaled and shown scaled to the Tracking window (Green dots) and the yellow corresponds to the center of the tracking window (Smaller Blue dot)

For both methods, OpenCV pre-defined classifiers [3] are used:

1. Frontal Face Classifier: haarcascade_frontalface_alt.xml

2. Eye Classifier: haarcascade_eye.xml

This minimizes the work rate of creating from scratch.

5. Results and Discussion

5.1 Algorithm 1: Eye Tracking via Range of Pupil Movement

The first algorithm for eye mouse tracking via range of pupil movements had failed due to mainly an overall unstable algorithm.

Firstly, when detecting a small pixel change via pupil movement, it caused a huge on-screen movement. This is due to scaling properties from the camera to the tracking screen. With the initialized pupil values being very small and scale ratio quite high, little to no movements caused inaccurate and continuous tracking points.

In a 2x2 pixel grid pattern, it still gave acceptable results as the overall area for each of the four regions (Top and Bottom Left and Right corners) was immense, hence any small irregular movements still placed the eye movements within the region. However a 3x3 pixel grid pattern gave minor but now incorrect positioning of the eye movements with regards to where the user was looking to. With every +1/-1 pixel change in real life was a huge difference on the tracking window that was decreasing in region area.

Secondly, reflection from bright surfaces, e.g. white tracking screen caused inaccurate pupil detection. Because the process of detecting pupil involved locating the darkest part of the eye via Integral imaging and Haar cascading features, using a bright screen caused a reflection onto the pupil area that caused the loss of dark areas when initializing it hence often detected different/multiple parts of the eye and this led to problem discussed directly above. Using a black screen window instead of a white helped minimize the problem for both Method 1 and Method 2 algorithms.

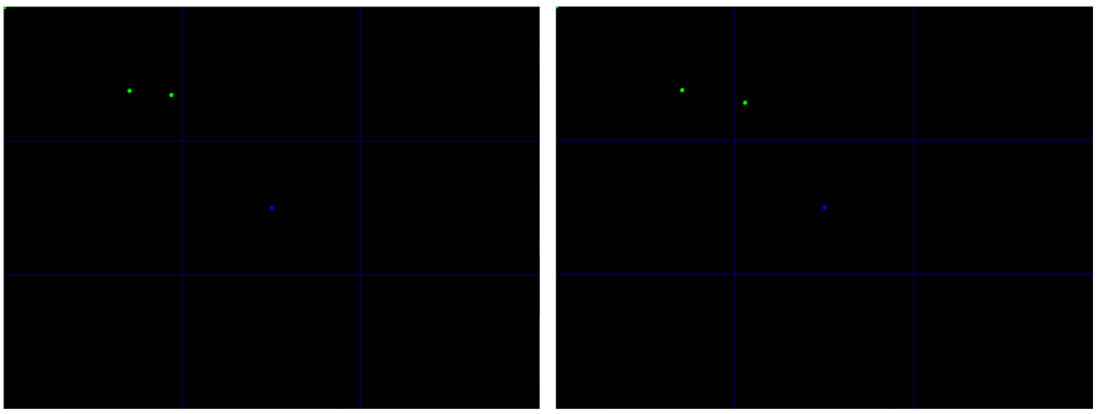


Fig: Pupil tracking is unstable despite no movements in real time in a 3x3 grid pattern

5.2 Algorithm 2: Eye Mouse Tracking via Face Location

As displayed from results below, with increases in the grid pattern, there are decreases in the accuracy and increases in error ratio.
2x2, 3x3 and 4x4 grip patterns had a 0% error ratio showing the algorithm was 100% accurate in detecting and tracking a users' pupil movement. Also shown in the graph, from 8x8 to a 9x9 grid pattern there is a greater error ratio than accuracy(Table 1 & Figure 1).

For the subsequent testing which tested for the stability of the patterns within regions, results showed first grid sizes from 2x 2 to 6 x 6 grids that there was 100% stability meaning the parts of the region the user had carefully placed their eyes on, the pointer was stable and did not move outside of grid boundaries hence Maximum of 4 seconds met. This showed persistent accuracy throughout the different grid levels. It however decreases after 6x6 grid pattern as the region areas become smaller and crowded and this increases likelihood of points slightly moving of the regions(Table 1 & Figure 1).

For both these test, the level of accuracy reached was encouraging. In general, it managed to get high accuracy level throughout most of the grid patterns with just a few less accurate results for more complex grid levels. Certain limitations that may have caused a later accuracy decrease in the testing may be due to scaling problems as from method 1. It cannot be fully solved as the uncertainties in scaling from real time views to a general standard 1280 x 960 screen requires real time advanced calculations which can be cumbersome. It may also be due to human errors, slight or considerable head movements and different eye – head directions. The program only tests for face location hence which ever direction the face is directed to, the eyes must follow. Regardless, the accuracy given for the test cases is promising.

Compared to Dynavox, this is still quite acceptable considering the camera used was not of high standard and the pre-defined classifiers in the OpenCV library were somewhat basic and perhaps required further improvements.

Test 1: Eye Mouse Accuracy Test

Table1: Accuracy Test for increasing Pixel Grid Pattern

Grid Pattern	Total Boxes	Correct Tracking	Wrong Tracking	Accuracy Ratio	Error Ratio
2x2	4	4	0	100.00%	0.00%
3x3	9	9	0	100.00%	0.00%
4x4	16	16	0	100.00%	0.00%
5x5	25	23	2	92.00%	8.00%
6x6	36	28	7	77.78%	19.44%
7x7	49	35	14	71.43%	28.57%
8x8	64	39	25	60.94%	39.06%
9x9	81	12	69	14.81%	85.19%

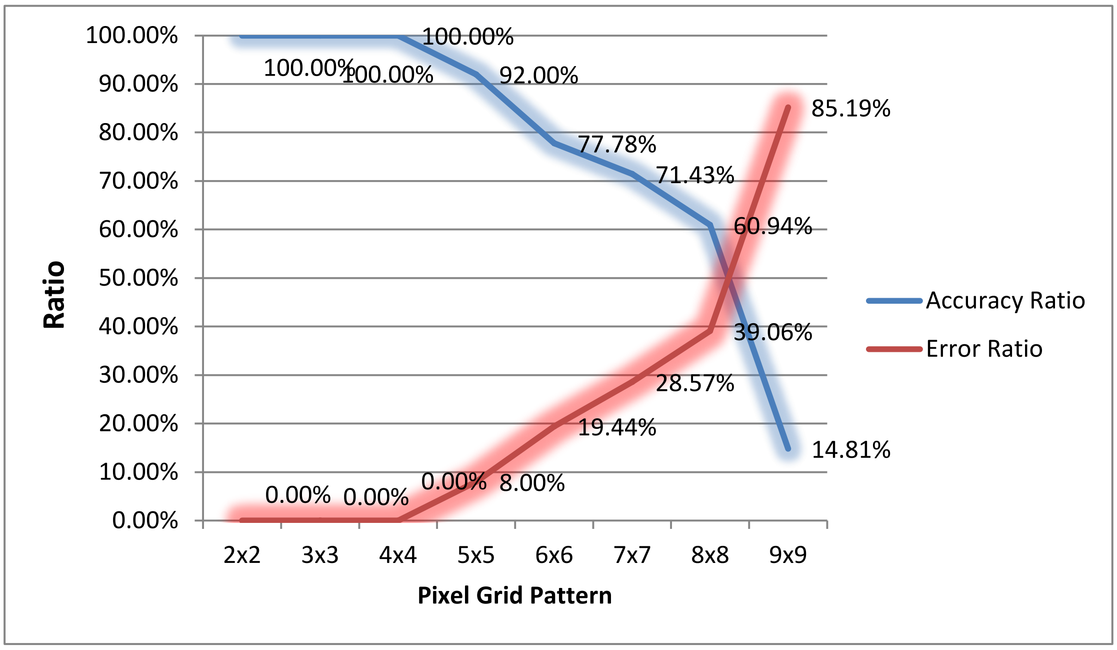


Figure1: Pixel Grid vs. Accuracy/Error Ratio Graph

Test 2: Region Accuracy and Stability Test

Grid Pattern	Stability Time(Max 4 seconds)	Accuracy Ratio
2x2	4	100%
3x3	4	100%
4x4	4	100%
5x5	4	100%
6x6	4	100%
7x7	3	75%
8x8	3	75%
9x9	2	50%

Table2: Testing for stability inside a certain region on Tracking Window

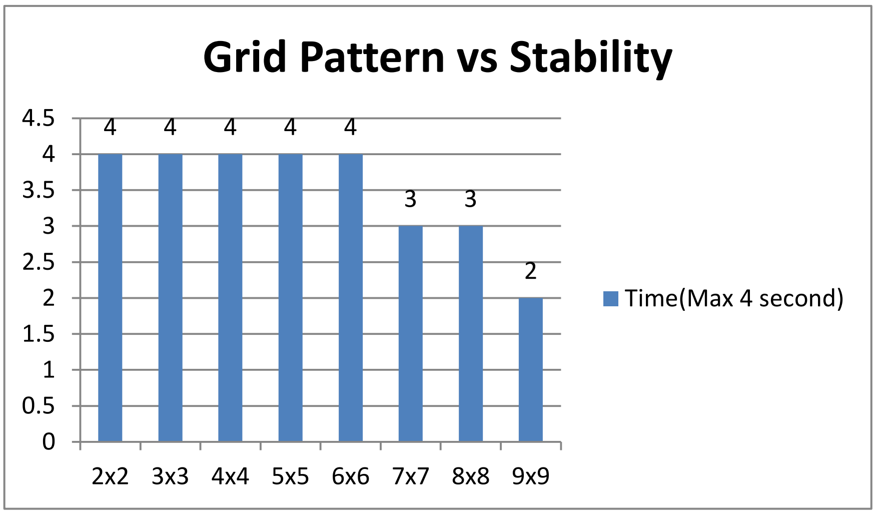


Figure2: Graph showing Grid Pattern vs. Stability timings for Tracking Window regions

6. Conclusion

To conclude, from observations and results collected, there is a reasonable amount of accuracy present despite using a non-high definition web camera and the standard pre-defined classifiers present in the OpenCV. User tests showed some accuracy in locating and tracking eye movement and passed most of stability tests for the regions it covered. However, more testing and development on the pre-defined classifiers and overall algorithm is needed so that it may be able to detect dynamic movements, not only static, allowing users more freedom and with high accuracy in reading faces and reducing limitations in pupil detection and movement.

Compared to Dynavox, it shares almost similar qualities in detecting faces however with smaller accuracy. The algorithm basically compromises between cost and accuracy of the tracking system and assuming further developments can be made to it, the future for it as open source software for use is promising. It may become a good testing ground for disabled and non-disabled users without access to an advanced eye tracking software or those with general interest in this particular field.

-

7. Bibliography

[1] Bradski, Gary R and Adrian Kaehler. *Learning Opencv*. Sebastopol, CA: O'Reilly, 2008. Print.

[2] Viola, Paul and Michael J. Jones. "Robust Real-Time Face Detection"*International Journal of Computer Vision* 57.2 (2004): 137-154. Web.

[3] "Cascade Classifier Training — Opencv 2.4.13.0 Documentation". *Docs.opencv.org*. N.p., 2016.Web. 26 June 2016.
http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html

[4] "Codeblocks". *Wiki.codeblocks.org*. N.p., 2016. Web. 26 June 2016.

[5] "Dynavox". *Wikipedia*.N.p., 2016. Web. 26 June 2016.
https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_
framework.