Camera Image Based Method of Real Time Gaze Detection and Interaction

Tobias Brächter, Dietmar Gerhardt
Department of Electronics and Signal Processing

DOI: 10.29322/IJSRP.10.11.2020.p10777
http://dx.doi.org/10.29322/IJSRP.10.11.2020.p10777

Abstract- There are many possible applications for tracking the gaze of a person. For example, it can be used to analyze which areas of a website are attracting the most attention in order to optimize its layout. Another possible application is creating a barrier-free computer interaction for people with a limited ability to move. Some companies are specialized in gaze tracking technologies and offer different solutions. The downside of those solutions is, that they require special hardware and therefore are not affordable for everyone. This paper presents a method using only one camera for gaze tracking. It is based completely on free accessible software and only needs an ordinary camera, which is integrated into almost every laptop or can be connected as external hardware. The target is to proof that it is possible to obtain a usable gaze interaction while using simple hardware and algorithms. By this method it is possible to achieve resolutions of up to 17 columns and 15 rows, which is enough for special applications.

Index Terms- computer vision, gaze detection, gaze interaction, gaze tracking, machine learning

I. Introduction

There are many possible applications for tracking the gaze of a person. These can be divided into passive applications, which only gather information and do not interfere with the user experience, and active applications, where the user can utilize them as an input device. Passive applications are often used for marketing reasons, for example to investigate which areas of a website or product attract the most attention of the user and use this information to design a better layout. But safety related applications exist as well, for example in the automotive sector, where eye detection is used to ensure that the driver is focused on driving, as described in [1] and [2]. Active applications on the other hand aim to create a better user experience by providing an alternative or additional input device for the user. The range varies from making a computer usable for physically disabled persons, e.g. for text input as described in [3], over increasing productivity in different scenarios ([4], [5], [6], [7], [8]) up to using it to control the camera in videogames as described in [9]. A software used to emulate mouse input from gaze tracking input is presented in [10]. Investigations on how gaze interaction performs as input method against the commonly used computer mouse are shown in [11].

Different companies, for example Tobii, have specialized themselves on this topic and provide different gaze tracking solutions based on hardware specifically developed for this purpose, where different techniques are used to receive high accuracy. Most of them are camera based, whereas other approaches also exist, such as contact lenses with integrated inductors, which create a magnetic field that can be measured. An overview of often used techniques is collocated in [12]. Camera based solutions tend to use setups with multiple cameras to obtain a depth of information and compensate distortion caused by viewing angles and for other reasons. When it comes to detecting pupils, infrared lighting is often utilized to detect pupils in the first step and to determine their gaze direction in the second step. The reason for this is that a certain, very small area of the pupil reflects infrared lighting and therefore can be easily detected by a camera. The location of this area relatively to the whole pupil can be used to precisely determine the gaze direction. Another approach is presented in [13], where several light sources at different locations are used along with raytracing to determine the gaze direction. Those solutions deliver a high quality, but the downside is the usage of specialized hardware, that has to be purchased and may not be affordable for everyone.
The motivation for this work is to proof that it is possible to create a free gaze tracking solution that is only realized in software and works with ordinary webcams in the first step. Furthermore, the accuracy of the developed software will be tested.

II. Approach

The main objective is to determine the location on the computer screen where the user is looking at, in the following referred to as target location. Basically, two kinds of information are needed, the first one is the location of the eyes relatively to the computer screen and the second one is the gaze direction.
A best-case scenario would be if this information would be known exactly. In this case the gaze direction vectors could easily be projected onto the computer screen and after that the point of intersection could be determined, which would be the target location.
The reality is a lot more difficult. The only information available is the two-dimensional frontal image of the user's face, which is captured by a camera that ideally is in a fixed location relatively to the computer screen. Otherwise there would be additional insecurities to deal with. From that image

the eye locations have to be determined as well as the gaze direction. A value that cannot directly be derived from that image is the location of the screen or its size, the easiest approach to tackle that problem is to calibrate the system, which will be described in section IV.
The approach to achieving the main goal can be divided into two main tasks. The first task is the detection and localization of the relevant facial structures that are needed and will be described in section III. The second task is to determine the target location based on the information received from the first task and will be described in section IV.

III. Detection of eyes and pupils

As described previously, the first task is to gain the relevant facial structures and their locations from the image received by the camera. One structure needed is the location of the eye within the image. The gaze direction cannot be gained directly. But it can be derived indirectly from the location of the pupil. As the pupil is the part of the eye, which absorbs the light, it is directed to the location the person is looking at. Therefore, you can determine the gaze direction indirectly by determining the location of the center of the pupil relatively to the location of the eye. In conclusion, the information that has to be derived from the image are the locations of the eyes and pupils.
Dlib is a cross-platform software library written in C++ that offers the needed algorithms and is used to detect and locate the eyes, which is done in three steps, which are the following:
1. Detect and locate the face in the image
2. Detect and locate facial key points
3. Derive the eye locations from the facial key points
These algorithms are based on machine learning, where large datasets, e.g. the one described in [14], are used to train neural networks in solving the addressed problems.

To detect the face, the frontal face detector provided by *Dlib* is used. The detector needs a grayscale image as input and returns a list of objects, where each object represents a face found in the image and provides information about it, such as the border locations. The camera image is captured and converted into a grayscale image using OpenCV, an open source computer vision and machine learning software library.
The first face from the list returned from the detector is used, assuming there are no other faces detected in the image, which never caused any trouble while only one person was present in the image.
In the next step, the facial key points are derived using the shape predictor provided by *Dlib*. To create a shape predictor, a trained database has to be downloaded first, which will then be provided to the shape predictor. The used database is a database with 68 relevant facial landmarks and can be downloaded at *http://dlib.net/files/*. After the shape predictor is created, it can be given the grayscale image of the face and the face object provided by the frontal face detector and will determine and return the locations of those 68 facial landmarks, which then can be used for further processing.

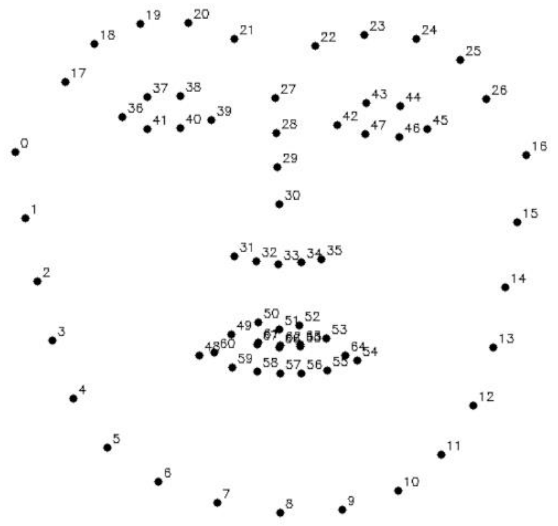


Figure 1 Eye detection by the landmarks 36, 39, 42 and 45 of the shape predictor provided by Dlib

Figure 1 shows the facial landmarks returned by the shape predictor. The next task is to derive the eye locations from those landmarks. To reach that goal, the landmarks 36, 39, 42 and 45 are used, which present the left and right outward locations of the respective eye. In an approach to approximately determine the center of the eye, the mean location of these two landmarks is calculated as shown in Figure 2, which is not exactly the center, but the best result to receive from this data. The upper and lower landmarks given for each eye are not used, as they mark the eyelids and therefore move if the user opens or closes his or her eyes, whereas a fixed location of the eye is wanted.

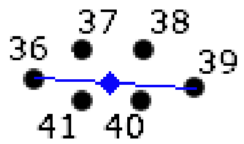


Figure 2 Location of the eye center based on the left and right landmark of the shape predictor

To this point, the implementation is straightforward and based on the free accessible toolkit *Dlib*. However, the detection of the pupil is a difficult task and has to be implemented manually. In the first step, the section containing the eye is extracted out of the camera image. As the pupil is built to absorb light, its brightness is usually the lowest compared to the surrounding area of the face, which makes the brightness to a possible property to filter by. Through experimenting it was discovered that the red channel of an RGB-image led to the best selectivity for the pupil against the surrounding area. Therefore, in the next step a grayscale image from the extracted section is created by only using the red channel. As the pupil should be the darkest area in the image, a binary filter is used, which turns every pixel in the image above the given threshold to white, every pixel below the threshold to black. Theoretically only the pixels belonging to the pupil are the ones that turn black, if the threshold is set correctly.

Practically there are same obstacles, for example bad or irregular lighting will lead to problems.

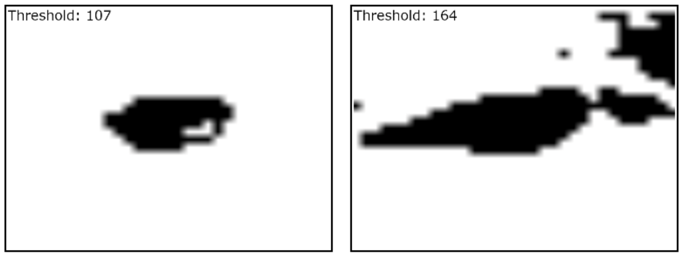


Figure 3 Impact of threshold on pupil detection

Figure 3 shows for example, how the threshold affects the accuracy of the pupil detection. On the left, the pupil is filtered out good, on the right the selected threshold is too high and other surrounding dark areas remain in the image, too. To select the proper threshold, the application starts with a threshold of zero and increases it, until a certain area is detected. The user can change the threshold manually if the result is not satisfying enough. To determine the center of the pupil, the center of the remaining area after filtering is calculated and used as location. As long as the lighting is good and the pupil is not covered, this approach works pretty well, if the simplicity of the algorithm is considered. Figure 4 shows a screenshot of a demo for the face detection.

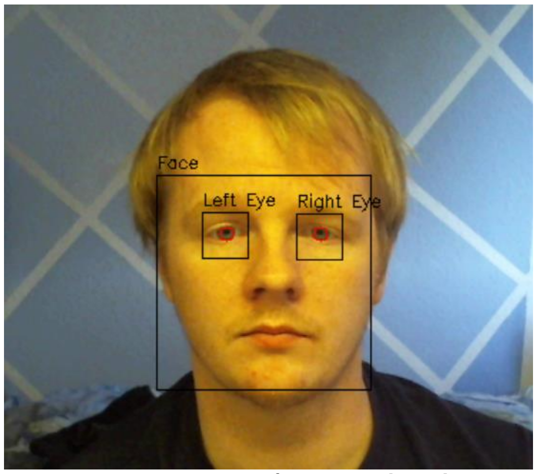


Figure 4 Face Detection of a person without glasses

Figure 5 shows another screenshot that demonstrates that the detection also works with (borderless) glasses.

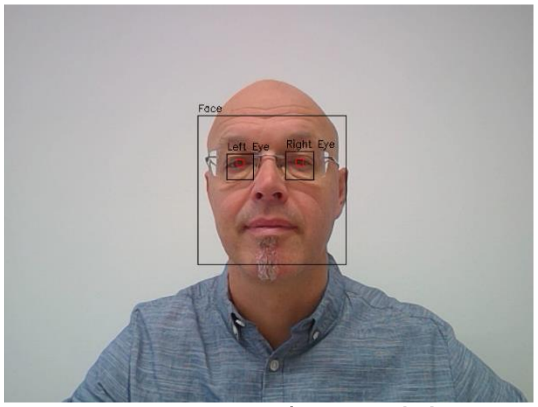


Figure 5 Face Detection of a person with glasses

IV. Determining the Target Location

After the detection and localization of the eyes and pupils are achieved, the target location has to be determined. As described in the previous sections, the location of the eyes and the gaze direction have to be determined in order to achieve that.

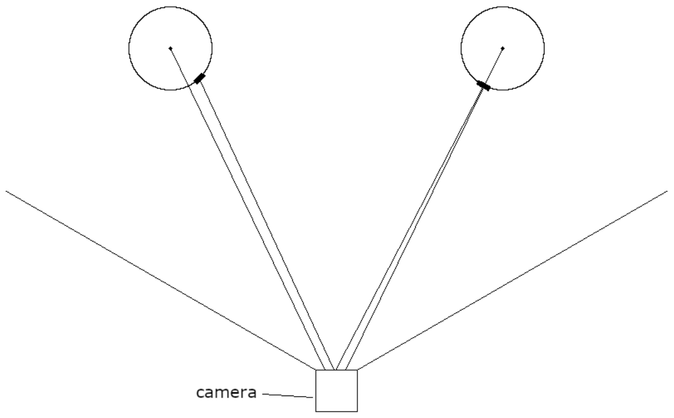


Figure 6 Projection of the centers of the eyes and pupils on the camera

Figure 6 shows how the reality gets captured by the camera. Using only one camera it is necessary to develop a projection model of the gaze direction in the computer screen. A main problem that occurs is, that the image contains no information about the absolute size or distance of the objects in it. Only the relative size of different objects to another within the image can be determined. As the approach is expected to be as simple as possible, these problems are not addressed directly and are handled by the calibration routine, which is described later in this section. There is also an impact by perspective distortions, which should usually be corrected by a fitting model.
The main goal, which is to realize gaze tracking by using an ordinary webcam, is also covered in [15], where neural networks were fed with images of the eyes in order to determine the target location. The authors in [16] and [17] also use neural networks to determine the gaze direction from the image and describe problems occurring with this approach, as for example bad generalization. In contrast to neural networks, this work aims to determine the target location by extracting specific values from the given image and feed them to a predefined model.

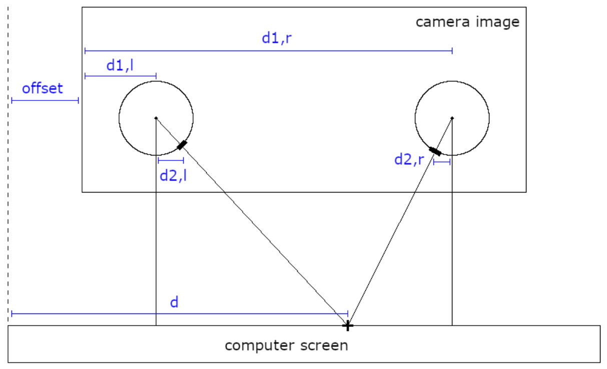


Figure 7 Gaze model to determine the target location

Figure 7 shows the model used to determine the target location, illustrated as a top view, only showing the x-axis. Determining the location of the eye seems rather simple, but it has to be taken into account, that various effects lead to the circumstance, that a movement of the eye in the camera image is not equal to the movement relatively to the computer screen in pixels. To keep the model simple, it is assumed that the movement in the camera image is proportional to the movement in the reality. According to the intercept theorems and the shown model, the distance between the pupil and the center of the eye is proportional to the distance between the location and the eye location projected on the screen. Further a constant was added to compensate occurring offsets, e.g. the offset between the camera image and the computer screen. To take both eyes into account, the target point is determined separately for each eye and finally calculated by using the mean value of the both target points. This model leads to the following equation per eye and per axis:

$$d = \alpha \cdot d_1 + \beta \cdot d_2 + \gamma \qquad \textit{Eq. 1}$$

Where *d* is the target location (x or y), d_1 is the location of the eye (left or right) and d_2 is the distance between the pupil center and the eye center both in the corresponding axis. The coefficients *α*, *β* and *γ* have to be determined through calibration. What should be mentioned, is, that the algorithm assumes that the distance between the user and the computer screen does not change. If it would change, the used coefficients would have to change accordingly, which cannot be implemented with simple approaches.
The calibration process is rather simple. The application shows several markers on the screen one after another. The user has to look at the marker and push a button to confirm that he or she is looking at the marked location. When the button is pressed, the current face detection data is appended to a list along with the shown screen location and the next marker is being shown. After the predefined locations were shown, the three coefficients *α*, *β* and *γ* for each eye and axis have to be determined. As the model assumes that the solution is a linear equation with two independent variables d_1 and d_2, the three coefficients *α*, *β* and *γ* can be determined by using the multiple linear regression. A linear regression model can be imported from the *sklearn* package and therefore must not be implemented manually. The model receives a list of vectors of several inputs along with the corresponding target locations in the function called *fit*. The target locations are drawn onto the computer screen, so the user looks at them while calibrating. After receiving those values, the linear regression model tries to determine the coefficients while using the least squares method to minimize the error over all input values. The calibration can be done several times if wanted, which would result in expanding the list of input values and the corresponding target locations and therefore minimizing statistical errors. The software creates four different linear regression models, one per every eye and every axis. After calibration, the models are given the input values received from the face detector continuously, so that a target location is determined for each eye separately. The final target location is then determined by calculating the mean value of both target locations.

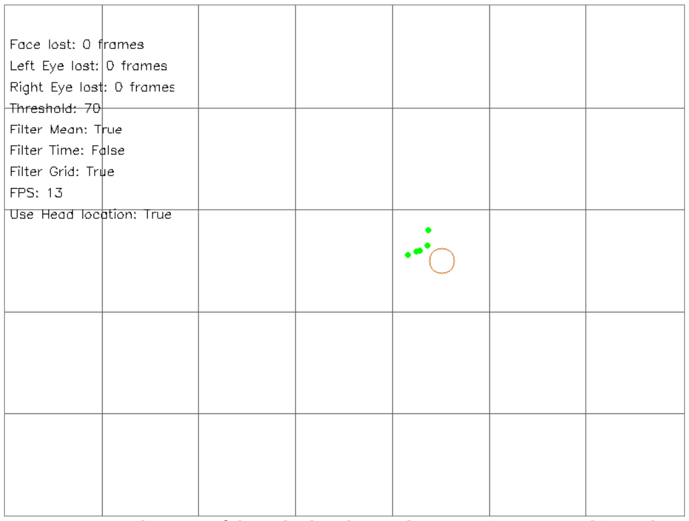
V. Filters

To optimize the results of the algorithm, different filters were implemented, which can be switched on and off independently. Without any filters, the algorithm already is able to determine a certain area around the location the user is looking at, but that area is relatively large and the cursor tends to shake randomly within this area. The different filters aim to solve this problem.
One filter is the mean filter, which saves the determined screen location of the last 5 frames and calculates the mean value of them, therefore representing a floating mean filter. This filter increases the accuracy while maintaining a high responsiveness.
Another filter called time filter tries to calm the cursor by only moving it towards the new target location by a certain part of the distance instead of directly setting the new location. The increase of accuracy using this filter is low, while the responsiveness of the cursor is decreased noticeable.
The third filter is called grid filter. It divides the screen into a grid with at runtime changeable amounts of rows and columns. The cursor gets centered in the cell which contains the determined cursor location, so that the screen gets divided into a finite number of cells which can be focused separately. This filter can also be used as a benchmark for the accuracy that can be achieved, which is described in the next section.

VI. Testing the Algorithm

The minimum requirement is fulfilled if the user is able to generally move the displayed cursor in the general direction he or she is looking. The best case would be if the determined target location would match the real target location at any given time. To determine the achieved quality of the application, a test procedure has to be defined and executed by different users and under different circumstances.
The outcome of this work is an algorithm that should be able to be implemented easily on different systems. As the used programming language is *Python* and all hardware accesses are realized through generalized library functions, it can be run on all platforms that support *Python*.
In general, the algorithm works as intended, but a test procedure has to be defined to determine its quality. The grid filter is used for this purpose, along with the mean filter,

which increases the quality. It was defined that the user must be able to focus a specific cell for at least 2 seconds, which was assumed to be long enough to submit a selection in a software developed for this purpose. Starting with a rather small number of rows and columns, the number gets increased until the user is no longer able to focus on a specific cell. The highest number of rows and columns, where the user is able to focus every cell separately is used as the indicator for the achieved resolution. The number of cells that could not be focused at are recorded along with the cell number, named error cells. Figure 8 shows a screenshot of the test software while running a test.



```
Face lost: 0 frames
Left Eye lost: 0 frames
Right Eye lost: 0 frames
Threshold: 70
Filter Mean: True
Filter Time: False
Filter Grid: True
FPS: 13
Use Head location: True
```

Figure 8 Visualization of the calculated gaze direction (orange circle) on the computer screen using mean filter with the last 5 (green) points and grid filter

As described previously, the gaze model was intended to determine the target location by projecting the gaze direction onto the screen and adding it to the eye location. This approach was meant to make the algorithm robust against head movements. During first tests the impact of the head movement on the determined target location could indeed be minimized by calibrating several times with different head locations, but the general accuracy became very low, as the outer cells could not be focused anymore and the determined target location often received an offset along the horizontal axis. The low achieved accuracy led to the decision to only test the software with a fixed head location, meaning the user held his or her head still while testing. But it was discovered that movements of the eye location can be interpreted as changes in the face alignment, as it also is mentioned in [18]. A few tests confirmed that by still including the eye location in the model and tilting the face into the gaze direction, which also felt more natural, the accuracy could be increased. As the software is also meant to be used by people that possibly are not able to move the head at all, a gaze interaction only based on pupil movements should also be tested. Therefore, two different test cases were defined, the first one with no head movement at all, the second one with tilting the head into the gaze direction. Every user had to test both test cases in order to determine the usability for the given case. As the linear regression reacts very sensitively to head movements after the calibration was done without moving the head, the eye location in equation 1 was not used while testing the first case, which can be switched on or off in runtime.

The software was tested on two different systems. One system was a Microsoft Surface Book 2 with a 15 inch display running Microsoft Windows 10. The built-in camera was able to record in the resolution 1920x1080 pixels. The other system was a desktop with a 28 inch display and an external camera with a resolution of 1280x720 pixels, also running Microsoft Windows 10. By default, the camera image received through OpenCV was given in a resolution of 640x480 pixels. The resolution of the received image can be changed, but the main target was to test the software in different environments by different (looking) users, so that the default resolution was kept. Both systems reached a stable speed of around 10 to 15 frames per second without optimizing the code. That is fast enough for common eye-controlled applications that are not time critical in user interaction. In this case it can be seen as real time when the response time from the user input to an application via gaze control up to 100 ms is acceptable. The test results are shown below.

Table 1 Test results without head movement

User	**System**	**Columns**	**Rows**	**Error Cells**
1	Surface	7	5	0
2	Surface	7	5	0
3	Surface	7	5	22
4	Desktop	7	5	1
5	Desktop	7	5	1
6	Desktop	7	5	1

Table 2 Test results with head movement

User	**System**	**Columns**	**Rows**	**Error Cells**
1	Surface	17	15	0
2	Surface	12	10	0
3	Surface	12	10	0
4	Desktop	15	12	0
5	Desktop	15	12	0
6	Desktop	15	12	0

As can be seen, the test results without head movement already seem to be good enough to interact with a computer by using software that is specifically designed for it, except for user 3. It has to be mentioned, that user 3 was wearing small reading glasses with a thick border, which probably disturbed the pupil detection. The other users testing on the Surface Book reached a high amount of focusable cells. All users testing on the desktop system had trouble to focus one of the cells located in a corner, but otherwise the results were good.

By adding the eye location to the equation and tilting the head into the looking direction while calibrating and using the software, all users were able to receive results that were significantly better, especially user 3. In this use case, there were at least 120 cells focusable, user 1 even reached 255 focusable cells on the Surface Book, so it would be possible to integrate an exhaustive keyboard on the screen that could be gaze controlled, or even for example navigate and select objects on a website that is designed in a more coarse style.

VII. Discussion

In general, the test results correspond to some observations made while developing and testing the face detection algorithm. Determining the eye location was very stable and accurate, whereas the pupil location was often not centered correctly and tended to shake slightly, so that the eye location is the more reliable source to determine the target location. It can be assumed that the quality will increase further if the pupil detection quality gets improved.

While all users testing on the desktop system had very similar if not identical experiences, the users testing on the surface book experienced different accuracies. It has to be highlighted at this point, that the screen of the Surface Book is significantly smaller than the screen of the desktop system, therefore meaning that the same number of focusable cells indicate a higher accuracy on the Surface Book. User 1 achieved a very high resolution. User 2 gained a resolution that was not as high, where the impact of the pupil color (dark brown) could be relevant in this case and should be investigated in further studies. User 3 had massive problems focusing cells while only using the pupil detection, which could be caused by the small reading glasses with thick borders. In all cases, the achieved resolution was way higher and all cells were focusable when the head movement was included.

Caused by the lack of different test systems it can only be guessed which other circumstances may have had an impact on the results. First of all, both systems were tested at completely different locations with different light conditions, where it can be clearly mentioned, that the lighting has a significant impact on the quality of the results. Another impact could be the camera location: While the camera of the Surface Book is integrated directly and near the screen area, the desktop system used an external camera, which had to be placed in front of the screen. Another explanation could be found in the camera resolution. Even though the image received within the software had the same (low) resolution, it must have been downscaled at some point. The camera of the Surface Book records at a resolution more than 2 times of the resolution provided by the external camera used in the desktop system. At this point an effect could have had an impact similar to super sampling, a technique used to improve the image quality in videogames. Super sampling renders the image at a higher resolution than the screen is capable to show and downscales it afterwards, which leads to a better image quality. The same effect could have occurred in this case and therefore could have improved the detection.

The inaccuracy of the pupil detection can be explained by several causes. The first cause is, that the detection relies only on the brightness of the image, where it was assumed, that the whole pupil is the darkest area of the image. But the brightness of each pixel strongly depends on the lighting, which can be changed significantly by moving the head a little. Furthermore, the best threshold value is different for both eyes, but the effort to set different threshold values for both eyes is not worth it, because the best threshold value changes continuously with every small change in the lighting or head orientation. Providing homogenous lighting helped a lot in minimizing this. Also, the pupil reflects light in some areas, leading to some pixels within the pupil that are above the given threshold and therefore get filtered out. In conclusion the binary filter itself already creates an inaccuracy by leaving some pixels belonging to the pupil out while including others that are not belonging to the pupil. After that, another problem is, that the pupil is almost never completely visible in the upper and lower region of the eye, as it is more or less covered by the eyelids. Even if all, and only all of the pixels belonging to the visible part of the pupil were detected, this circumstance would lead to inaccuracies. A well-defined model of the pupil, that would be able to detect it based on other properties than just brightness and determine the location precisely based on the visible part of its shape should be able to deliver better results. But it has to be stressed at this point, that the target of this work was to realize gaze tracking by using an easy approach, which the previously described model would clearly not be.

Table 3 shows a sample of the determined coefficients according to equation 1 in order to verify that all of three coefficients α, β and γ have a significant impact on the determined target location. As the table shows, all of the coefficients differ noticeable from zero and therefore have a significant impact.

Table 3 Example results of the linear regression (with head movement)

Eye	**Axis**	α	β	γ
Left	x	-51.39	-168.04	17107.48
Left	y	104.97	114.14	-13932.21
Right	x	-56.71	-168.04	22537.34
Right	y	105.60	155.41	-13807.17

VIII. Conclusion

As mentioned in the introduction, the aim of this project was to proof that a solution for gaze interaction with a computer that does not depend on special hardware and therefore is accessible for nearly everyone can be developed. As the time available for this project was limited, the target was to achieve a proof of concept rather than a solution that competes in quality with the ones offered by companies that are specialized on this topic.

Generally, it can be said that the target could be achieved. Using the free library *Dlib*, machine learning based algorithms to detect the eye location could be implemented fast and delivered a stable and accurate solution. The algorithm to locate the pupil on the other hand is not that accurate and tends to shake, but considering the simplicity of it, the result is surprisingly usable. Furthermore, the values and correlations used to determine the target location seem to be well-chosen, as the screen can be divided into a lot of cells that can be focused separately. As the eye location itself as indicator for head tilting seems to be the main contributor for leading to a high accuracy, it can be assumed that the solution can be improved a lot by enhancing the quality of the pupil detection.

As the software is made up very simple, there are plenty possible improvements that could be implemented later. First of all, as already mentioned, the accuracy can probably be enhanced a lot by improving the pupil detection. Furthermore, the distance of the user to the screen could be taken into account as well by estimating it through the size of certain facial structures. Also it should be investigated if it is possible that a single calibration for each system is enough to set it up for future use by different users, or maybe it is even possible that no explicit calibration is needed at all to create an usable experience. Another aspect that was not covered in detail, is the resolution of the used image. Using a higher resolution, if it is available, should also improve the accuracy. Furthermore, the algorithm should be advanced to detect blinking or other gestures in order to receive a confirmation of the user to actually confirm a selection. Another function that could be implemented, is a compensation of head movement, which would allow the user to move his or her head around in a certain range without decreasing the accuracy, an approach to this is described in [19].
In conclusion the developed method for gaze detection and interaction works as a proof of concept and already delivers a good quality if the simplicity of its approach is considered. It is possible to realize gaze interaction by only using the image received from an ordinary camera. Also there seems to be much potential to improve the algorithm and the achieved quality, so that the project can be described as a success.

References

[1] T. S. Achudan, N. Gobinath, *Real Time Eye Gaze Detection Using Machine Learning Techniques*, International Conference on Signal Processing and Communications, 2016, Pages 1-6

[2] M. Q. Khan, S. Lee, *Gaze and Eye Tracking: Techniques and Applications in ADAS*, sensors Journal, 2019, Pages 1-3

[3] P. Majaranta, K.-J. Räihä, *Twenty Years of Eye Typing: Systems and Design Issuses*, Proceedings of Eye Tracking Research and Applications, 2002, Pages 1-10

[4] A. Çöltekin, J. Hempel, A. Brychtova, I. Giannopoulos, S. Stellmach, R. Dachselt, *GAZE AND FEET AS ADDITIONAL INPUT MODALITIES FOR INTERACTING WITH GEOSPATIAL INTERFACES*, ISPRS Annals of the Photogrammetry, 2016, Pages 1-8

[5] S. Stellmach, R. Dachselt, *Still Looking: Investigating Seamless Gaze-supported Selection, Positioning, and Manipulation of Distant Targets*, Changing Perspectives, 2013, Pages 1-10

[6] S. Stellmach, R. Dachselt, *Designing Gaze-Based User Interfaces for Steering in Virtual Environments*, ETRA Journal, 2012, Pages 1-4

[7] E. Castellina, F. Corno, *Multimodal Gaze interaction in 3D Virtual Environments*, Conference on Communication by Gaze Interaction, 2008, Pages 1-5

[8] R. J. K. Jacob, *Eye Movement-Based Human-Computer Interaction Techniques: Toward Non-Command Interfaces*, 2003, Pages 1-5

[9] L. Nacke, S. Stellmach, D. Sasse, C. A. Lindley, *Gameplay experience in a gaze interaction game*, Conference on Commuincation by Gaze Interaction, 2009, Pages 1-6

[10] S. Vickers, H. O. Istance, A. Hyrskykari, N. Ali, R. Bates, *Keeping an eye on the game: eye gaze interaction with Massively Multiplayer Online Games and virtual communities for motor impaired users*, 2008, Pages 1-8

[11] R. Bednarik, T. Gowases, M. Tukiainen, *Gaze interaction enhances problem solving: Effects of dwell-time based, gaze-augmented, and mouse interaction on problem solving strategies and user experience*, Journal of Eye Movement Research, 2009, Pages 1-10

[12] H. Singh, Dr. J. Singh, *Human Eye Tracking and Related Issues: A Review*, International Journal of Scientific and Research Publications, 2012, Pages 1-9

[13] C. Hennessey, B. Noureddin, P. Lawrence, *A Single Camera Eye-Gaze Tracking System with Free Head Motion*, symposium on Eye tracking research & applications, 2006, Pages 18

[14] S. Porta, B. Bossavit, R. Cabeza, A. Larumbe-Bergera, G. Garde, A. Villanueva, *U2Eyes: a binocular dataset for eye tracking and gaze estimation*, IEEE Conference on Computer Vision and Pattern Recognition, 2019, Pages 1-5

[15] W. Sewell, O. Komogortsev, *Real-Time Eye Gaze Tracking With an Unmodified Commodity Webcam Employing a Neural Network*, Proceedings of ACM Conference on Huma Factors in Computing Systems, 2010, Pages 1-6

[16] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matsuik, A. Torralba, *Eye Tracking for Everyone*, IEEE Conference on Computer Vision and Pattern Recognition, 2016, Pages 1-9

[17] K. Wang, R. Zhao, H. Su, Q. Ji, *Generalizing Eye Tracking with Bayesian Adversarial Learning*, IEEE Conference on Computer Vision and Pattern Recognition, 2019, Pages 1-10

[18] P. Wang, M. B. Green, Q. Ji, *Automatic Eye Detection and Its Validation*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, Page 1

[19] F. L. Coutinho, C. H. Morimoto, *Improving Head Movement Tolerance of Cross-Ratio Based Eye Trackers*, Springer Science+Business Media, 2012, Pages 1-23Authors

Authors

First Author – Tobias Brächter, student in Master of Engineering, tobias.braechter@hs-bochum.de
Second Author – Dietmar Gerhardt, professor, head of the Department of Electronics and Signal Processing, dietmar.gerhardt@hs-bochum.de
Institute – Bochum University of Applied Sciences, Campus Velbert/Heiligenhaus, Kettwiger Straße 20, D-42579 Heiligenhaus, Germany

Correspondence Author – Prof. Dr.-Ing. Dietmar Gerhardt, dietmar.gerhardt@hs-bochum.de, Bochum University of Applied Sciences, Campus Velbert/Heiligenhaus, Kettwiger Straße 20, D-42579 Heiligenhaus, Germany, Fon.: +49 2056 584816713