Development of a Computer Vision eye gaze tracking algorithm for Virtual Reality applications.

Leopold Hebert-Stevens
CentraleSupelec
Université Paris-Saclay

Adrien Loizeau
CentraleSupelec
Université Paris-Saclay

1. Abstract

In this paper, we present a comprehensive overview of techniques for eye detection and tracking in Virtual Reality applications. We review various eye models and detection techniques, and investigate the impact of different image pre-processing techniques on the accuracy and efficiency of our results. We focus on traditional image analysis methods that have been proven to achieve accurate results without requiring a large amount of data or computational resources. We evaluate our models using multiple datasets such as the Labeled Pupils in the Wild (LPW), UnityEyes and MIIP. Our results indicate that a combination of edge detection, dilation and thresholding is an effective method for improving the accuracy and robustness of the circle detection algorithm for pupil segmentation. Furthermore, we found that the Determinant of Hessian is the best blob detection technique. Our study also highlights the potential use of synthesized eye images like the UnityEyes dataset, but the limitations of not being able to generate images ourselves. The MIIP dataset provided a unique advantage as it provides already normalized images taken in front of the subject, which reduces problems of head inclination and normalization. Overall, our goal is to provide a comprehensive overview of the techniques for eye detection and tracking and identify the most promising methods for achieving robust and efficient results in real-world applications.

2. Introduction

Eyes are one of the most salient features of the human face and their movements play an important role in expressing a person's desires, needs, cognitive processes, emotional states, and interpersonal relations. The importance of eye movements to the individual's perception of and attention to the visual world is implicitly acknowledged as it is the method through which we gather the information necessary to navigate our surroundings and identify objects. Robust non-intrusive eye detection and tracking is, therefore, crucial for the development of human-computer interaction, attentive user interfaces, and understanding human affective states.

The unique geometric, and motion characteristics and complexity to control the environment of the eyes make the eye detection and gaze estimation a very challenging task and has been a main research research topic for many decades. Futhermore, the commercial usage of gaze estimation has majorly been security applications such as biometric recognition [1] which did not helped the domain to expand much. Nevertheless, these past years things have been changing since the rise of VR and Augmented Reality headsets, many leading tech leading companies like Meta put all their efforts in creating the future of this new reality. Among all the challenges of actual headsets, one of the major ones is having enough computational resources to give a realistic experience to the user. Eye segmentation and gaze detection could play a major role here by allowing resources only to what the user is focusing on, thus improving the quality of that segment and providing a more realistic interaction for the user. Our goal during this project is to explore the simplest ways of doing both eye segmentation and gaze estimation for Virtual Reality (VR) application.

To achieve this goal, we will provide a comprehensive overview of the techniques for eye detection and tracking, and identify the most promising methods for achieving robust and efficient results in real-world applications. With the increasing popularity of virtual and augmented reality, the development of accurate and efficient eye detection and tracking methods is more important than ever. To do we will in a first part present the related work we used to have a better understanding of the state of the art techniques. Then we will precisely detail the different methods we tested for both problems. Once we have found the best approaches we will evaluate the models on various datasets to determine which ones are the most accurate. Finally we will conclude our project with an overall view of our work.

In this report we won't investigate the use of deep learning techniques for eye detection and tracking as these techniques require too much computational resources.

3. Related work

Eye detection is a challenging task due to factors such as occlusion of eyes, variability in their size, reflectivity, and head pose. Thankfully as this problem has been on researcher's mind for a long time, numerous techniques have been tested out and referenced since the first researches [7]. To refer to this domain the keywords are "eye detection", "eye tracking" and "gaze estimation".

Among the different papers, Hansen's survey [2] gave a complete overview of the different techniques available in 2010, this paper is perfect for us as at that time deep learning wasn't existing. This paper teaches us many taxonomy of eye detection methods including shape-based, appearance-based, and hybrid methods and also on gaze detection. Let's have a quick overview of those ones.

3.1. Eye and pupil detection

Shape-based methods typically rely on a geometric eye model and a similarity measure. Simple elliptical shape models, such as those used to track only the iris or the pupil, have been used in the past. These models often use voting-based or model fitting techniques, such as edge detection and the Hough transform, to locate the eye. However, these simple models are not capable of capturing variations and interactions of eye features such as eyelids, eye corners, and eyebrows.

Complex shape models, such as the deformable template model, attempt to improve upon the limitations of simple models by using more detailed modeling. However, these methods are computationally demanding and may require high contrast images and close initialization to the eye. They also struggle with handling face pose changes and eye occlusions.

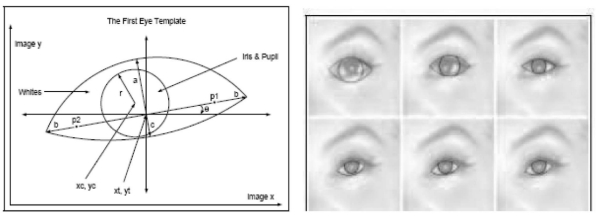

Figure 1. Complex shape models example and results

Feature-based shape methods explore the characteristics of the human eye to identify a set of distinctive features around the eyes. These methods include local features by intensity, local features by filter responses, and pupil detection.

Appearance-based methods detect and track eyes directly based on the photo-metric appearance, such as the color distribution or filter responses of the eye and its surroundings. These methods are less sensitive to illumination changes and can be implemented using techniques such as support vector machines, hidden Markov models, and principal component analysis

3.2. Gaze estimation

Early gaze tracking systems used three type of models [5]:

- the ones based on points such as center of the pupil or glint
- the ones based on shapes
- and hybrid models combining points and shapes

The point ones are the most interesting as they use an IR light source to improve contrast and obtain stable gaze estimation results, something totally achievable in a VR headset. The idea here is to found the inclinaison of the eye and the head thanks to the IR light and therefore be able to found the angle of the iris. If this method seems simple, it however makes the erroneous assumption that the corneal surface is a perfect mirror, which can lead to inaccuracies in gaze estimation. To compensate for this, researchers have proposed polynomial regression methods to model the correlation between pupil centers.

One example of this is the method proposed by Merchant et al. in 1974 [3], which used a real-time video-based eye tracker employing IR light, with a collection of mirrors and galvanometers to allow for head movements. They used the pupil-glint vector and a linear mapping to estimate the point of regard, and noticed nonlinearities with large pupil-glint angles, which they compensated for using polynomial regression.

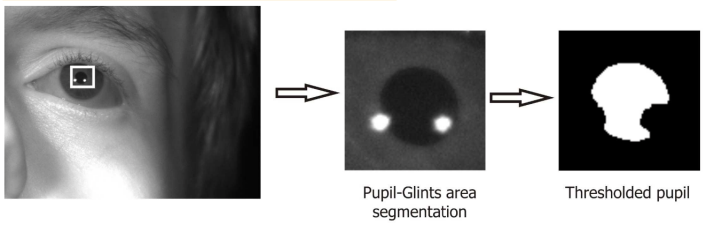

Figure 2. Example of glint based method

Similarly, Morimoto et al. [4] also used a single camera and utilized a second order polynomial for the x and y directions separately to represent a direct mapping of the glint-pupil difference vector to the point of regard. However, this method had the drawback that the calibration mapping decayed as the head moved away from its original position, which we can not do in our study.

Many methods exist in the literature today for gaze estimation but few of them are easily applicable for a student project. In fact as we will see later in the report, most of

the datasets presented in the papers are not easily accessible or easily configurable. In traditional computer vision, the ideal situation for our study would be having a total control on the data we have to perfectly calibrate our datas. As this isn't possible we will only focus in this study on possible gaze estimation techiques with the few datasets we had available.

4. Methodology

4.1. Segmentation of the eye

Segmenting the eye is an essential step in the process of gaze estimation. This technique involves isolating the eye region in an image, which enables the analysis of the eye independently from the rest of the image. The position of the pupils, being the center of the eye, is crucial for determining the point of regard and thus the gaze direction. The pupils are usually the most salient features in the eye and are easy to detect, also the pupils are relatively small and dark compared to the rest of the eye, which makes them an ideal feature for segmentation tasks. To detect pupils in the eye, the two methods used are circle and blob detection.

4.1.1 Image transformation

Despite the fact that pupils are relatively easy to detect due to their darker appearance compared to the surrounding area of the eye, the segmentation algorithm may still encounter challenges. In particular, the algorithm may incorrectly identify eyelashes as part of the pupils due to their similar circular shape. This is particularly true when make up was on such as mascara, as it can further accentuate the eyelashes appearance in the edge detection, and make them more difficult to distinguish from the pupils.

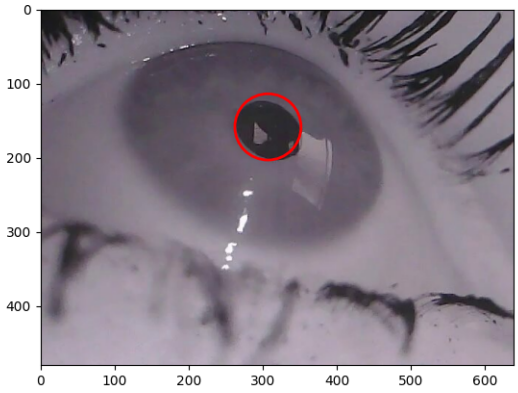

Figure 3. Example of an image where the eyelashes are more intense due to make up (mascara).

To address this issue, various image transformations are applied to the input image to enhance the robustness of the detection algorithm.

Both the circle and blob detection methods were evaluated on a variety of transformed images, with the goal of identifying which image transformations produced the most accurate and precise results for each method.

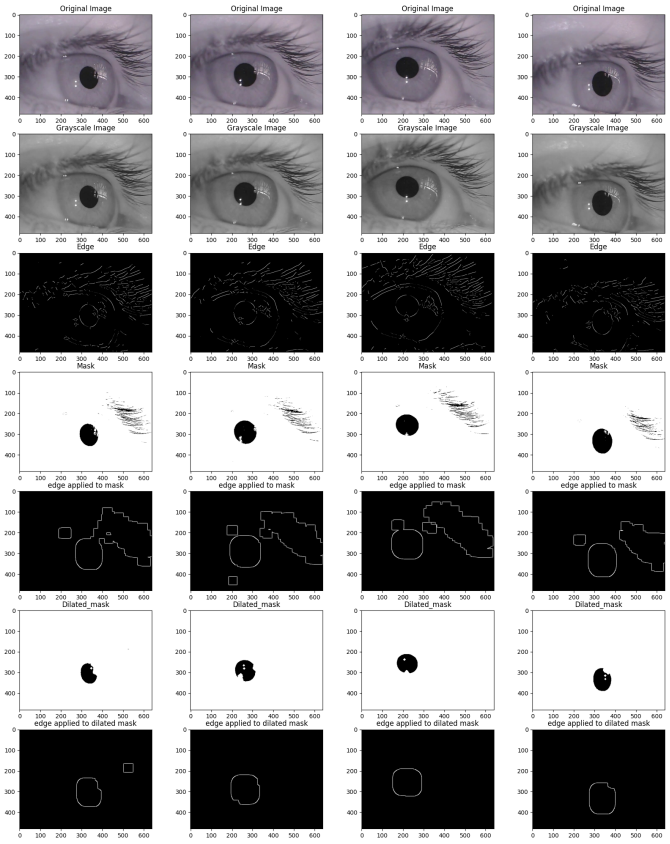

Figure 4. Different transformations applied to 4 randomly selected images of the dataset. From top to botom: Original image, grayscale filter, Edge detector applied to grayscale image, mask applied to grayscale image, Edge detector applied on threshold image, Dilation apply to threshold image, Edge detection applied to dilated thresholds image.

These transformations aim to improve the contrast between the pupils and the surrounding areas of the eye, making it easier to accurately segment and detect the pupils. For example, applying dilation to the threshold image removes the eyelashes and gives a better edge detection result. Therefore, the overall performance of the algorithm is improved, and the risk of false positives is reduced.

4.1.2 Circle detection

In order to accurately detect the pupils, we implemented a circle detection algorithm. Despite the possibility of the pupils might appears elliptical on certain images due to variations in camera inclination, light reflects, and eye position,

this algorithm operates by identifying circular shapes within the eye region of the image.

The steps of our algorithm are as follow:

- Convert the input image to grayscale. This is done to simplify the image processing and reduce the amount of noise in the image.
- Smooth the grayscale image to reduce noise using a Gaussian filter.
- Apply a threshold to create a binary image, keeping only the area of interest.
- Apply dilation to the binary images to increase the size of the object and making the edges more pronounced; while smoothing the image by removing small isolated pixels and filling small holes.
- Apply the Canny edge detector developped during the previous assignment to find edges.
- Iterate over different radius values in the range of min_radius to max_radius. For each radius value:
  - Create a binary mask of a circle with the current radius value.
  - Use the mask to extract the circular region from the edges image.
  - Count the number of edge pixels in the circular region.
  - If the number of edge pixels is greater than the threshold, record the coordinates of the center of the circle and its radius.
- Return the list of circles with their center and radius
- Draw the circles on the original image

Upon testing our circle detection algorithm on a variety of transformed images, we realised that the algorithm performed best when applied to the edge image of the dilated threshold image. The results show that the estimated coordinates and corresponding circles were significantly more accurate and reliable when compared to other transformed images. This suggests that edge detection in combination with dilation and thresholding is an effective method for improving the accuracy and robustness of the circle detection algorithm for pupil segmentation.

However, as we will see in the results section, the execution time of these functions is very high. Therefore some libraries from Skimage were implemented, especially for edge detection and Hough transform to detect circular shapes.

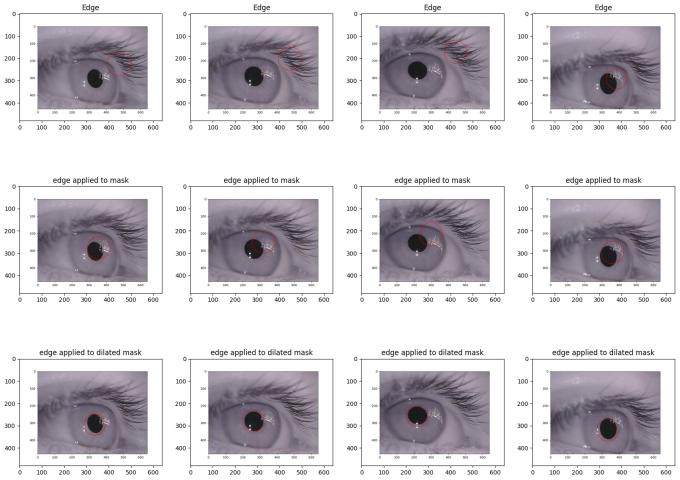

Figure 5. Circles detection algorithm tested on multiples transformed images.

4.1.3 Blob detection

Blob detection is a technique used in image processing to identify and locate connected regions of pixels that share a common property, such as color, brightness, or texture. These connected regions are known as ”blobs.” Blob detection algorithms are used to identify and locate the blobs in an image, and to extract information about their properties, such as size, shape, and position. The advantage of using blob detection is that it can be used to detect blobs of any shape, not just circles. There are three main methods to detect blobs in an image:

- Laplacian of Gaussian (LoG) method, also known as scale-space blob detection, which uses the Laplacian of Gaussian function to detect blobs. The Laplacian of Gaussian function is a smoothed version of the Laplacian operator, which is sensitive to edges in an image. The extrema of the LoG function are then thresholded to detect the blobs.
- Difference of Gaussian (DoG) method, which uses the difference between two Gaussian filtered images to detect blobs. The method works by subtracting a low-pass filtered image from a high-pass filtered image. The resulting image will have high intensity values at the location of blobs.
- Determinant of Hessian (DoH) method, which uses the determinant of the Hessian matrix to detect blobs. The Hessian matrix is a square matrix of second-order partial derivatives of an image. The determinant of the Hessian matrix gives the rate of change of the blob's area over scale. The method works by thresholding the determinant of the Hessian matrix to detect blobs.

Similarly than for circle detection, taking an excessive

amount of time to compute and find the blobs in the image, we decided to go with the blob detection libraries from Skimage.

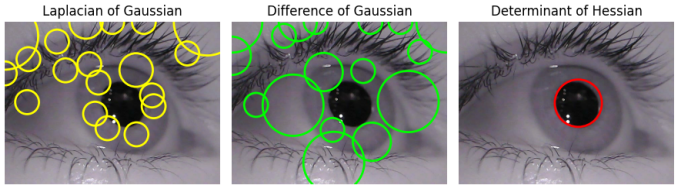

Figure 6. Different technique of blob detection.

From figure 6 it appears that the best blob detection technique is the Determinant of Hessian. Therefore we developed, as a proof of concept, a Difference of Hessain blob detector from scratch that works as follows:

- Convert the input image to grayscale.
- Apply a threshold to the grayscale image.
- Apply dilation.
- Compute the Hessian matrix of the grayscale image. The Hessian matrix is a square matrix of second-order partial derivatives of the image intensity function, which is used to detect blobs in the image.
- Compute the eigenvalues and eigenvectors of the Hessian matrix. Eigenvalues represent the magnitude of the blob, while eigenvectors represent the orientation of the blob.
- Set a threshold on the eigenvalues to remove small or insignificant blobs.
- Identify blobs by extracting connected components in the image that have eigenvalues above the threshold.
- Return the center of mass and the radius of each blob
- Draw the circles on the original image

Similarly than for circle detection, the blob detection algorithm was tested on multiples transformed images. It appears that the algorithm produce better results when applied to the threshold image, dilated or not.

4.2. Gaze prediction by machine learning

Once the pupils are segmented and the coordinates of the center of the pupils are found, it is possible to predict the gaze using a dataset that contains the coordinates of the center of the pupils and the corresponding gaze directions. The dataset is used to train a machine learning model, to learn the relationship between the pupils' coordinates and the gaze directions.

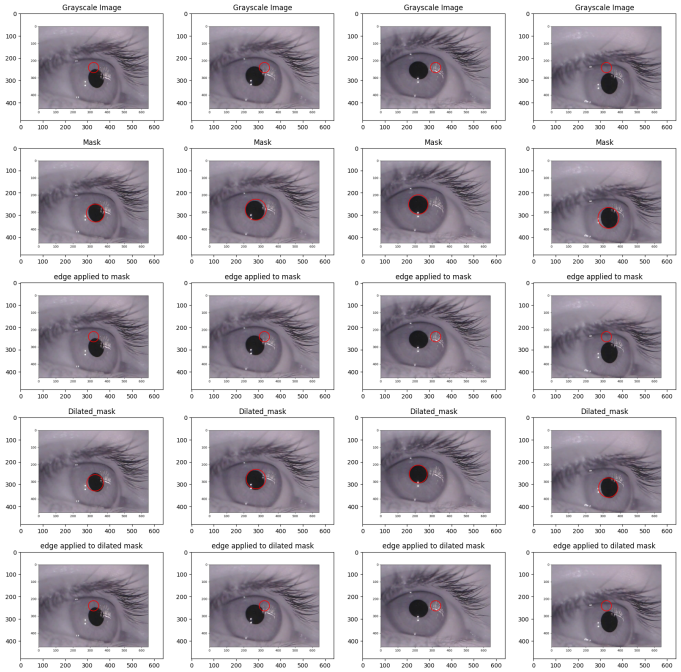

Figure 7. Blob detection algorithm tested on multiples transformed images.

In this case we used the multi-output regressor from scikit-learn library. Multi-output regressor is a type of regressor that can handle multiple input (x,y coordinates of the pupils) and output (x,y,z gaze coordinates) variables. Moreover, many types of regressor have been tested in the multi-output regressor model:

- RandomForestRegressor(): This is an ensemble regressor that builds multiple decision trees and combines their predictions to improve the overall accuracy. Random forest is a robust model and can handle high dimensional and non-linear data. It's also less prone to overfitting and can handle missing data.
- LinearRegression(): This is a simple linear regression model, it assumes that the relationship between the input and output variables is linear. It's a fast and efficient model that works well with small datasets.
- Ridge(): This is a regularized linear regression model, it adds a penalty term to the linear regression model to prevent overfitting. It can handle multicollinearity and improve the model's generalization.
- Lasso(): This is also a regularized linear regression model, it adds a penalty term to the linear regression model to prevent overfitting, but it differs from Ridge in that it is able to shrink some of the coefficients to zero, which can be useful in feature selection.

- DecisionTreeRegressor(): This is a decision tree-based regressor, it builds a tree-like model of decisions and their possible consequences. It's easy to interpret and visualize, but it can be prone to overfitting.
- SVR(): Support Vector Regression is a type of support vector machine (SVM) which uses the same principle as the SVM classifier, but it is used for regression tasks.

After the training process, the model can then be used to predict the gaze direction from the pupils' coordinates in new images.

4.3. Gaze approximation by geometrical approach

the geometrical approach to estimate the gaze from the coordinates of the pupils involves finding the vector that connects the pupils and the eyes center. This vector represents the gaze direction in the image plane. The angle of this vector in the image plane can be used to calculate the gaze direction in the x and y direction.

The vector that connects the pupils and the eyes center can be represented mathematically as:

$$V = (x_2 - x_1, y_2 - y_1)$$

Where (x_1, y_1) and (x_2, y_2) are the coordinates of the two pupils. This vector represents the gaze direction in the image plane.

Once we have the gaze vector, we can calculate the angle of this vector in the image plane. This angle can be represented in two dimensions, the x-direction (horizontal direction) and y-direction (vertical direction). The angle in the x-direction is the angle between the gaze vector and the x-axis. Similarly, the angle in the y-direction is the angle between the gaze vector and the y-axis.

The angle of the gaze vector in the x-direction can be calculated using the arctangent function:

$$\theta_x = atan2(V_y, V_x)$$

Where V_y and V_x are the y and x components of the gaze vector respectively.

The angle of the gaze vector in the y-direction can be calculated using the arcsin function:

$$\theta_y = asin(V_z/|V|)$$

Where V_z is the z component of the gaze vector and $|V|$ is the magnitude of the gaze vector.

Once we have the angle of the gaze vector in the x and y direction, we can use it to estimate the gaze direction in the 3D world coordinates. To do this, we need to take into account the intrinsic and extrinsic parameters of the camera such as the focal length, the principal point and the position of the camera.

5. Evaluation

5.1. Dataset tested

In our project proposal, we presented many datasets available for segmenting eyes and gaze tracking. However, it was challenging to find relevant datasets for our research. The images should be of high resolution and good lighting conditions. They should also be well-annotated with accurate information about the gaze direction, head inclination and eyes exacts position. Moreover, the images should be taken from a consistent angle and viewpoint to minimize variations in the data, which is important to ensure the generalisability of the algorithm. Here are three datasets we used:

5.1.1 Labeled Pupils in the Wild (LPW):

This dataset contains multiple videos of eyes of different individuals filmed inside a VR headset while looking in different directions. The dataset includes participants of different ethnicities, in both outdoor and indoor settings and includes different variants, such as subjects wearing makeup, contact lenses, and glasses. These videos come with annotations of the centre of the pupils tracked along the video. Transformed into frames, this dataset represents 12,000 images to train and tune the parameters of our eye segmentation and gaze-tracking algorithm.

In our work this dataset was the first one we used, it helped us to create our eye segmentation algorithms. But once we started working on the gaze estimation, we where quickly limited by the way the data was gathered. In fact, as the images were taken on different angles and on different light conditions, without any landmark (IR light for example) we started to look at a more suited dataset.

5.1.2 UnityEyes

The UnityEyes dataset[6] is a collection of synthesized eye region images created using a novel method that combines a generative 3D model of the human eye with a real-time rendering framework. The images are based on high-resolution 3D face scans and use real-time approximations for complex eye materials and structures, as well as anatomically-inspired procedural geometry methods for eyelid animation.

As presented in Viviane Clay's paper on Eye tracking in VR ([6]), using an engine to create the dataset seems to be a very useful method as the user have the full control on the environments, cameras and therefore have a very precise calculation of the gaze. Samples of the dataset are presented in Fig 3.

if using this dataset seemed to be our ideal solution, we quickly realised that we waren't able to generate images ourselves as we are not working on compatible computers.

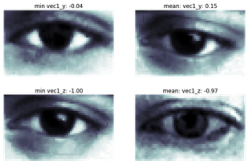

Figure 8. MIIP normalized dataset sample

The only solution we had was to use the already sampled examples, and this was a problem as the dataset used was providing eye images from various angles and thus was making the gaze estimation harder.

5.1.3 MIIP

The last dataset we used to test out our algorithms on was the MIIP dataset [8] that contains 213,659 images we collected from 15 participants during natural everyday laptop use over more than three months. This dataset is significantly more variable than existing ones with respect to appearance and illumination but has the unique advantage of providing already normalized images taken in front of the subject. These unique images are a goldmine for us as head inclination and normalization aren't problems anymore and we can therefore apply simpler methods for gaze estimation. Example of the normalized dataset is available at Fig 8.

5.2. Results

The eye segmentation and pupils coordinates estimation using both the circles and blob detector have been tested on the LPW dataset, and here are the following results:

	from scratch	Skimage library
Circle detect.	$\approx$ 30min	$\approx$ 2min05
Blob detect.	$\approx$ 25min	$\approx$ 1min50

Table 1. Execution times of different algorithms on 200 frames to segment and estimate the coordinates of the pupils

As explained in the methodology section, we developed circle and blob detection algorithms to segment and estimate the coordinates of the pupils in the eyes images from scratch. While these algorithms were effective in achieving our objectives, they were found to be relatively slow compared to pre-defined functions from computer vision libraries such as the Skimage library. As a result, for the rest of our study, we choose to use the optimised algorithm provided by the Skimage library in order to ensure efficient and fast pupil segmentation and gaze estimation. This decision was made to ensure the feasibility and scalability of our proposed method, as well as to minimize computational costs.

From Figure 9, we can notice that both the circle and blob detection have strengths and weaknesses. For example,

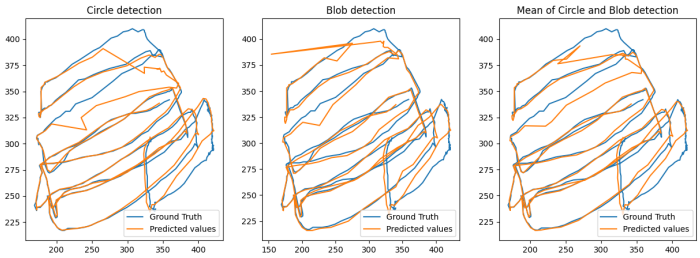

Figure 9. Comparison of estimated pupil coordinates using Circle Detection, Blob Detection, and Mean of Circle and Blob detection algorithm against the ground truth for a video of an eye looking in different directions.

the estimation from the circle detection is accurate but not precise as it is not very close to the ground truth but presents no outliers. On the other hand, the estimations based on the blob detection algorithm are very accurate but less precise. The predictions are very close to the ground truth but sometimes present anomalies. A clear example can be seen when the x-coordinate estimation is approximately 150 instead of 250. In order to mitigate these limitations and improve the robustness of our estimates, we propose a method that combines the strengths of both algorithms by taking the mean of their predictions.

Moreover, we can see that for position of the pupils very high or very down right, both algorithm methods struggle to detect and estimated correctly its position.

Metric	Circle	Blob	Mean
MAE	5.77	5.00	5.11
MSE	92.2	93.1	64.6
RMSE	9.62	9.65	8.03
R2	0.975	0.980	0.982

Table 2. Metrics applied to the same data than figure 9 for Circle, Blob and mean of both methods pupils estimation.

From the results presented in Table 2, it is clear that the combination of both methods (circle detection and blob detection) yields the most robust and accurate estimates of the pupils coordinates. This is likely due to the fact that the two methods have different strengths and weaknesses, and by combining them, the algorithm is able to compensate for their respective limitations.

The results suggest that the algorithm is able to achieve a high degree of accuracy and precision in estimating the coordinates of the pupils in the image, which is a key step in gaze estimation.

The same detection algorithms were also tested on the Unity Eye dataset, and here are some examples of the results.

Unfortunately, our detection algorithms were less accurate compared to other datasets in segmenting the eye and correctly estimating the center of the pupils. Hence, the re-

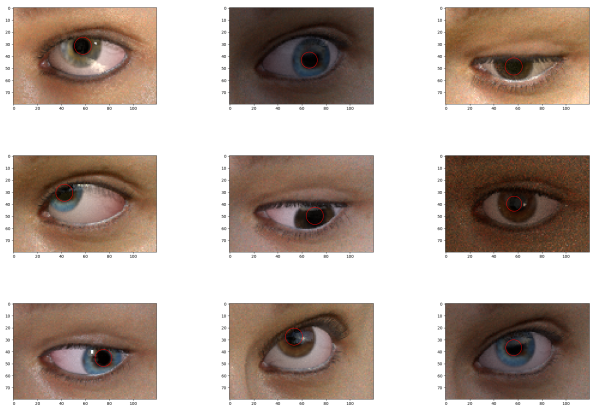

Table 3. Example of images form the unity_eye dataset, segmented by circle detection

sults obtained from this dataset are inconclusive. Further investigation is needed to fully understand the cause of this discrepancy and to improve the performance of our algorithm on this dataset.

Our machine learning algorithm's performance is heavily dependent on the accuracy of the input data. Due to the inaccuracies in our detection algorithm's estimation of the pupil coordinates, the gaze direction predictions generated by our machine learning model are very biased.

While the bias in the estimated coordinates may have affected the overall performance of our gaze estimation, a train-test split evaluation of our machine learning model on true data yielded promising results, suggesting that the model has the capacity to generalize well and produce accurate predictions of gaze directions based on pupils coordinates, as shown on table 4.

	MAE	MSE	RMSE	R2
Rand.For.Regres.	0.093	0.024	0.155	0.69
Lin.Regres.	0.155	0.035	0.188	0.410
Ridge	0.155	0.035	0.188	0.410
Lasso	0.17	0.045	0.211	0.324
Dec.Tree.Regres.	0.109	0.04	0.201	0.484
SVR	0.153	0.035	0.186	0.473

Table 4. Metrics for different regressor used in the multi-output regression algorithm from the train-test split evaluation of true data

6. Conclusion

In conclusion, this study aimed to explore the simplest ways of performing eye segmentation and gaze estimation for Virtual Reality (VR) applications. We examined various eye models and reviewed eye detection techniques that utilize these models, without utilizing deep learning techniques, as they require significant computational resources. Through our examination of traditional image analysis methods, we found that edge detection in combination with dilation and thresholding is an effective method for improving the accuracy and robustness of the circle detection algorithm for pupil segmentation. Additionally, the Determinant of Hessian (DoH) method was found to be the most effective for blob detection.

However, it is important to note that deep learning solutions have also shown to be effective for eye detection and tracking, and should not be disregarded as a potential solution for real-world applications. The increasing popularity of virtual and augmented reality highlights the importance of accurate and efficient eye detection and tracking methods, and further research in this area should consider the integration of deep learning techniques in addition to traditional image analysis methods.

References

[1] B. Adegoke, E. Omidiora, F. A.s, and J. Ojo. Iris segmentation: A survey. *International Journal of Modern Engineering Research (IJMER)*, 3:1885–1889, 01 2013.

[2] D. Hansen and Q. Ji. In the eye of the beholder: A survey of models for eyes and gaze. *IEEE transactions on pattern analysis and machine intelligence*, 32:478–500, 03 2010.

[3] J. Merchant, R. Morrissette, and J. L. Porterfield. Remote measurement of eye direction allowing subject motion over one cubic foot of space. *IEEE Trans. Biomed. Eng.*, 21(4):309–317, July 1974.

[4] C. Morimoto, D. Koons, A. Amir, and M. Flickner. Pupil detection and tracking using multiple light sources. *Image and Vision Computing*, 18:331–335, 03 2000.

[5] A. Villanueva, J. Cerrolaza, and R. Cabeza. *Geometry Issues of Gaze Estimation*. 10 2008.

[6] E. Wood, T. Baltrušaitis, L.-P. Morency, P. Robinson, and A. Bulling. Learning an appearance-based gaze estimator from one million synthesised images. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research Applications*, pages 131–138, 2016.

[7] A. Yarbus. Eye movements and vision. *Plenum Press*, 3, 1967.

[8] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. Appearance-based gaze estimation in the wild. jun 2015.