The Banny method for Eye Tracking

Marco Monteverde
DIBRIS
University of Genoa
Genoa, Italy
marcomonteverde1@gmail.com

Vasilii Tregubov
DIBRIS
University of Genoa
Genoa, Italy
tregubovvg@gmail.com

Abstract

In this report, we propose a method to perform eye tracking with an architecture that uses, MTCNN, Canny algorithm and HoughCircles. We also provide experimental results that show challenges we still have to address, and ideas to solve them in order to obtain an implementation strong enough to be used for tracking scenarios.

1. Introduction

The field of computer vision, focused on determining the direction of a person's gaze, has been rapidly advancing in recent decades due to significant commercial interest from numerous IT companies [1]. Eye tracking algorithms are employed to assess website attractiveness, test new games, and address various tasks in customer satisfaction analysis and also in medicine [2]. The evolution of this field is closely tied to advancements in neural networks, with most modern algorithms involving convolutional neural networks (CNNs) [3, 4, 5]. The core of any eye tracking algorithm is the ability to recognize a person's pupil and determine its direction in the image. Generally, two approaches are used: developing a deep CNN from scratch, which requires training on a large dataset of labeled data, or utilizing an existing neural network (e.g., ResNet, VGG16, EmotionNet, MTCNN) as a base and either fine-tuning it or adding an architectural superstructure for precise pupil detection. Training a model from scratch requires significant time and memory resource. Effective training needs a vast number of images with labeled eye pupils, and the model may struggle with blurry images where the cornea is hard to distinguish from the pupil. In this work, we develop an algorithm based on a backbone model, enhanced with a specialized architecture for determining gaze direction in images, and then apply this approach to videos.

2. Methods

The task of eye tracking on a computer screen can be solved by creating and combining several parts:

1. An algorithm that computes the corneal direction vector (eye vector) relative to the eye in the image.
2. A calibration procedure that maps the eye vector to the point on the computer screen at which the eye is pointing to.
3. An way to represent videos as sequences of images of a given frequency.

The final algorithm computes the path of the view on the screen, by applying (1) and (2) to

each image in the sequence obtained by (3). The algorithm works best under the following conditions:

1. The face, except for the eyes, is motionless during video recording.
2. The face is located on the vertical axis of image symmetry, so that the eye axis is perpendicular to the vertical axis, as well as to the normal vector from the screen plane.
3. The face has both eyes.

2.1. Pupil detection

For pupil detection we propose a new approach we call "Banny" (Backbone + Canny). Since in the framework of our problem we need to determine the pupil direction of a human face, we propose an approach (Fig.2.1) in which:

1. A pre-trained multitasking model is used as backbone of the algorithm, detecting several informative zones (e.g., eyes, nose, and mouth) on human face at once. It is used to compute the pseudo centers of two eyes.
2. Based on the statistical knowledge of facial proportions, the extended ROIs of the two eyes are computed.
3. The Canny algorithm is applied to the obtained ROIs, which is used to determine the left and right extremes of each eye, thus placing the eye in the center of the image.
4. The “schemes” of the eyes obtained by Canny, are fed as input to the HoughCircles algorithm, which detects circular patterns in the image (corneas).
5. The view vector for each eye is found as the difference between the center of the pattern and the calculated corneal center.
6. The final vector is found as the mean vector of the two eyes, or one of them if the second one is not detected.

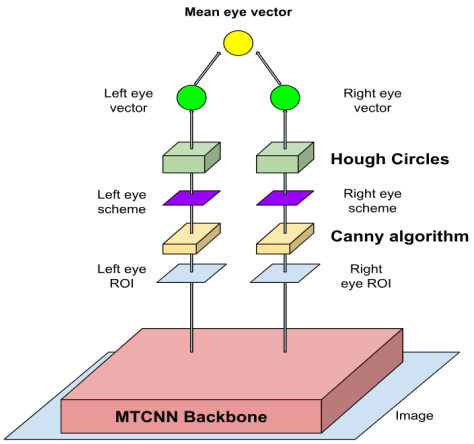

Figure 2.1: Architecture of the Banny algorithm.

The reason for using multitasking models is that they allow the detection of two eyes in an image at once, as well as the triangulation of a rotated or tilted face. The latter task is beyond the scope of our project: however, the usage of such a model easily achieves this functional extension. We have chosen the MTCNN model as the multitasking model for this project.

2.2. Proportions of the human face

After detecting the pseudo-center of the eye, we determine the boundaries of the expanded rectangle containing the eye image. The fact that a person may be at different distances from the camera when recording a face should be taken into account when calculating the ROI bounds. It is possible to do it on the

basis of knowledge about human face proportions, such as:

1. The average distance between the eyes of a human being is 63 millimeters.
2. The average diameter of the human eye is 24 millimeters.

Knowing the distance between the pseudo eye centers in image d, we can determine the ROI boundaries using the average proportions of (1) and (2) with a slight expansion to avoid losing potentially important eye boundary information:

$$\begin{cases} b_x = (24+6)/63 \times d, \\ b_y = (24+1)/63 \times d. \end{cases}$$

2.3. Calibration

For the calibration, we decided to create images consisting in a black screen and red dots in specific positions of the screen (one dot per image, as chown in fig.2.2), in order to take a photo of the user looking at the dots in fullscreen mode with the computer camera. This approach was performed manually by the user, but it can in future be automatized. For this work, we used 9 points in specific points of the screen.

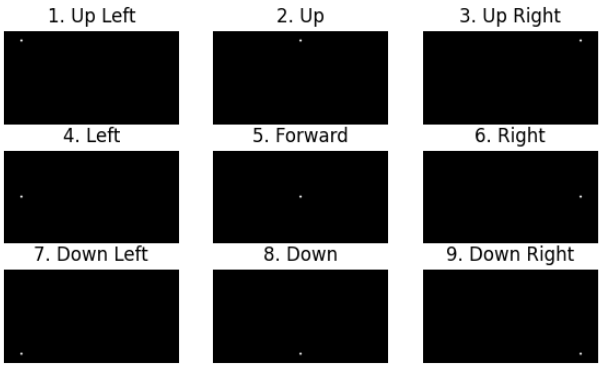

Figure 2.2: the nine calibration images.

2.4. Eye tracking

The tracking is performed taking a video of the user looking at the screen (in the same head position of the calibration) and applying the pupil detection algorithm in loop on each frame of the video. Then, black images with a green dot are created in the same way the calibration images were, but with the data obtained by the detection. These images are created in order, so that they can be recomposed in a video showing the trajectory of the gaze on the screen.

3. Experiments

In this section, we want to demonstrate the results of experiments with our model and analyze them in two parts: in the first, we describe the experiments with the gaze direction detection algorithm in images (Banny), and in the second, we describe the results of experiments with eye tracking in a video.

3.1. Banny algorithm

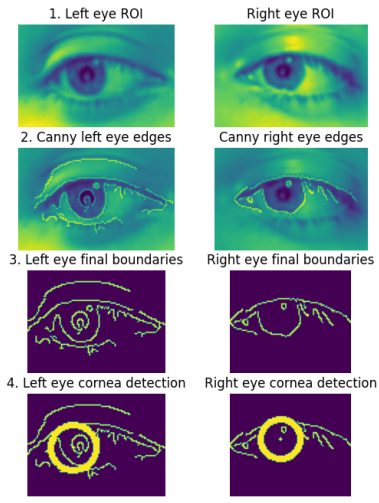

Figure 3.1: Steps of calculating pupil position from ROI.

First of all, we would like to demonstrate the steps of the Banny algorithm on some image. As an input image we have a photo of the face of one of the report authors looking to the left. The primary MTCNN model marks the pseudo center of the eye around which the ROI is created and then the Canny algorithm is applied, centering and finally calculating the corneal center. The steps described are visualized in fig.3.1.

Experiments with different eye directions (Fig.3.2) demonstrate the excellent ability of the algorithm to detect the horizontal eye direction, this shows the effectiveness of the horizontal image centering method. However, the detection of the vertical position is not so accurate, as can be seen by comparing the results in fig.3.2 and the table with the calculated coordinates of the eye vector direction.

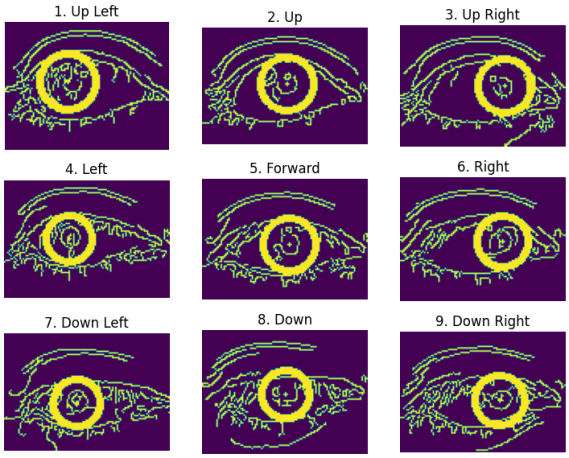

Figure 3.2: 9 diiferent directions of the eye and Banny algorithm results.

	Left	**Forward**	**Right**
Up	[12,3]	[-1,1]	[-14,0]
Forward	[12,0]	[-3,-3]	[-12,-1]
Down	[7,-7]	[0,-1]	[-11,-5]

Table 3.1: Estimated view vectors.

The insensitivity of the algorithm to the vertical direction arises because in this algorithm we do not perform vertical centering in any way, because it cannot be performed using the boundaries of the Canny scheme: if we limit the image to the eyelid width, the algorithm completely loses the ability to determine the "down" direction, because the gaze movement leads to eyelid narrowing and the cornea in such centering remains apparently strictly in the middle between them.

3.2. Eye tracking algorithm

First, it is worth to mention that the implementation of the algorithm resort to a Javascript script to extract the screen size of the user computer. This is due to our work being on Colab and the resulting impossibility to resort on python libraries that granted us direct access to the camera of the device. Due to the calibration being imprecise and not robust, the tracking we performed was very unstable and completely unreliable (Fig.3.3).

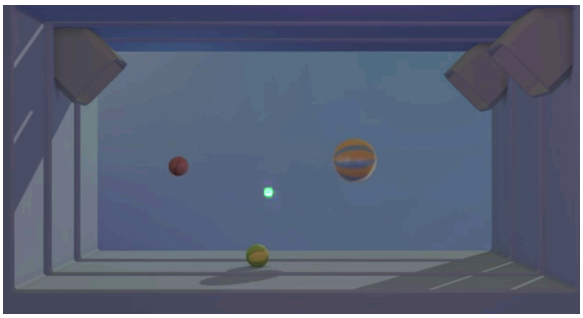

Figure 3.3: screenshot of the tracking result video. The gaze should be on the big ball. The full video (tracking_attempt.mp4) is provided with this report submission

In previous attempts, we had problems with the calibration being performed with a wrong camera and this resulted in tracking

results being out of screen, but this was fixed in later experiments. From the video, we can notice also that the horizontal movement is very sensitive while the vertical one is too insensitive: this is a direct consequence of the inability of the Banny algorithm to represent this type of movement.

4. Conclusions

We developed an eye tracking algorithm proposing our own approach to compute gaze direction, combining MTCNN, Canny algorithm and HoughCircles from the recorded video. The addition of Canny algorithm between ROI extraction and application of HoughCircles is our little "innovation". We found that this use of Canny algorithm significantly improves the ability of HoughCircles to accurately find the "circular regions" of corneas, since this eye schema does not contain noise reducing the accuracy of circle detection directly in the eye image. The tracking shows the need of a better way to compute the gaze: one possible choice is to track only one eye (this trick should alone be not sufficient, but it simplify the problem). Our experiments show that the algorithm needs further improvement: first, it should be modified to more accurately detect the vertical direction of gaze. For this purpose, triangulation can be used by adding a third external face point and calculate the relative position of the pupil. To achieve a smoother gaze movement, we can apply Kalman filtering. The calibration procedure also needs improvement, perhaps using trigonometric regression instead of linear for more accurate measure. Nevertheless, experiments suggest that there are prospects for using such approach in Eye tracking.

References

[1] K.Krafka*, A. Khosla*, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik and A. Torralba "Eye tracking for everyone" *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*(2016)

[2] Benedetta Franceschiello, Tommaso Di Noto, Alexia Bourgeois, Micah Murray, Astrid Minier, Pierre Pouget, Jonas Richiardi, Paolo Bartolomeo, Fabio Anselmi "Machine learning algorithms on eye tracking trajectories to classify patients with spatial neglect." *Computer Methods and Programs in Biomedicine. 221. 106929. 10.1016/j.cmpb.2022.106929* (2022)

[3] Nishan Gunawardena, Jeewani Anupama Ginige, Bahman Javadi, Gough Lui "Performance Analysis of CNN Models for Mobile Device Eye Tracking with Edge Computing" *Procedia Computer Science Volume 207, Pages 2291-2300* (2022)

[4] Jennifer, Joevian Krislynd, Steven Aprianto, and Derwin Suhartono "A Comparative Study of Various Convolutional Neural Network Architectures for Eye Tracking System" *Journal of Image and Graphics, Vol. 10, No. 4, December 2022* (2022)

[5] Jerry Lam, Moshe Eizenman "Convolutional Neural Networks for Eye Detection in Remote Gaze Estimation Systems" *Lecture Notes in Engineering and Computer Science. 2168.* (2008)

The project folder can be found at: drive.google.com/drive/u/0/folders/1_17oDwbOv3Rv9wNIB_PCcY4Z_gToUZ63