DETECTION OF EYE LOOK POISITION WITH COMPUTER VISION

Projet CSC51073 – 2025

3 décembre 2025

Nathan Duboisset - Roméo Nazaret

TABLE DES MATIÈRES

1 INTRODUCTION

1.1 Context and motivation

Computer vision and image processing are now widely used in many applications. Eye tracking and pupil detection have become important tools with practical uses.

Pupil detection systems have several useful applications. They can be used as anti-cheating systems in exams by monitoring where students look. They also help people with severe disabilities communicate through eye movements when they cannot move other parts of their body. These applications show why we need accurate and real-time pupil detection methods.

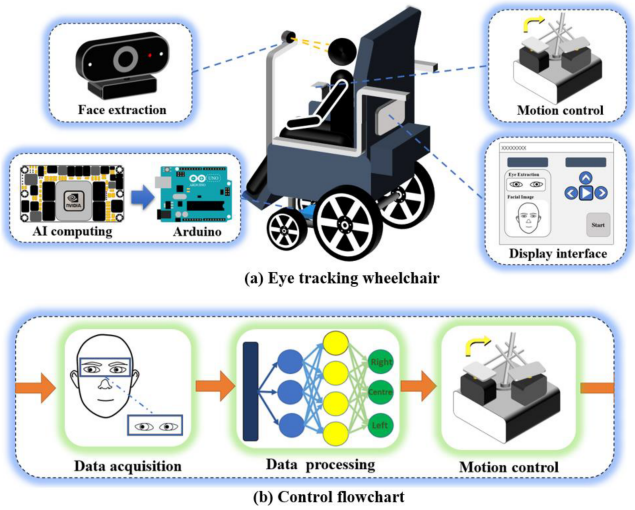

Figure 1 – Example of pupil detection application

1.2 Problem

Detecting the position of the eyes is a difficult task because the eyes can be occluded by other objects, the eyes can be closed, they can be looking in different directions, etc. Also there is a lot of technical problem : how to have precise description of the eyes position regardless of head position, quality of the video, number of images per second, etc.

We must be able to adapt to all these different situations while keeping a good precision and a low latency, thus using fast enough solutions.

1.3 Project goals

Our goal is to detect the position of the eyes in a video with a good precision and a low latency. We will use a computer vision algorithm to detect the eyes position. We will also try to use a machine learning algorithm to detect the eyes position. As we will use the webcam of our computer, we will have a difficult enough problem, as the quality of the video is not always good.

2

STATE OF THE ART

2.1 AFIG 2007

The AFIG 2007 paper by Raynal [1] presents a morphological approach for pupil detection. The method uses two coefficients : C_m (morphological quality) measures how well a candidate circle matches edge contours, and C_c (colorimetric quality) measures color similarity with the detected iris. The algorithm works in two passes : a coarse scan to find candidate circles, then a refinement step. This method is computationally efficient and does not require training data.

2.2 MediaPipe Face Mesh

MediaPipe is an open-source framework by Google that provides real-time computer vision solutions. Face Mesh detects 468 facial landmarks using a CNN-based model. The system uses a two-stage approach : first detecting the face bounding box, then applying a neural network to predict 3D landmark positions. It runs in real-time on standard hardware and works across multiple platforms. For eye tracking applications, we can extract the subset of landmarks around the eyes (about 32 points per eye) to isolate the eye regions before applying pupil detection algorithms.

2.3 Other approaches

Recent pupil detection methods include deep learning approaches using U-Net or similar architectures for segmentation, and hybrid methods combining classical computer vision with neural networks. Some systems use infrared cameras for better contrast, but this requires specialized hardware. Our project focuses on visible light cameras to keep the system accessible and low-cost.

3

IMPLEMENTED METHOD

3.1 General architecture

We chose to focus first on the detection of the eyes gaze position rather than detecting the position of the objects themselves as these tools are already widely used and have a lot of research, and good and fast implementations. Thus we used mediapipe and took the computing from there.

The main idea is then to use mathematical computations to estimate our positions, ensuring a good precision and a low latency.

3.2 Gathering of positions data

We use mediapipe to gather the positions of elements we need for this project. We use the face mesh algorithm to get the positions of the eyes and the face. We use the iris landmarks to get the positions of the pupils. We also use mediapipe to gather the position of the pupils on the face. Using this, we compute where should be the position of the middle of the eyeballs, and thus obtain two vectors supposed to be the direction of the eye gaze. We tried to use other methodsto gather the position of the pupiles on the face, but it revealed less efficient than the mediapipe one. The idea was to use direct machine learning, by first gathering images of the eye, taken using the webcam, by zooming on the eye, using the eye positions, and then use a convolution neural network to compute the position of the center of the pupil. The idea was to have for each coordinate x or y, n output "positions", then use a softmax to have a probability for each position, and then use the position with the highest probability. We had to use a loss for gradient calculation that took into account the fact that these outputs were not independant, and rather positionnal. This was not as efficient as expected so we also tried the following method. We thought it would be more efficient to use the geometric particularities of the eye and pupil to get these coordinates. By starting from the same images we had before("photo" of the eye relatively close), we "looked" for the pupile by trying different disks with different center and radius, and take the one fitting the best to the pupil. We had to score each disk, by taking into account that often, most of the pupil is covered by the eye, that people can have different eye color (because we mostly used the faceact that the pupil is a black element in the middle of a rather light eye), and that the pupil is not always a perfect circle. This approach work relatively well, but often placed the pupil on a "wrong circle" when it detected another circle pattern. It also was hard to find a good balance between trying too many disks hyperparameters(center and radius), and wanting to be fast enough. As a conclusion, we decided to use the mediapipe method, as it was more reliable, and also gave a position relative to the face, thus really helpful for us.

3.3 Detection of the position of the screen

Once we had the vectors modelizing the position of the eyes, we can intersect them with the expected position of the screen, and knowing the dimensions of the screen, its resolution etc. we can compute the position of the gaze on the screen. But to have that computed, we need to know the distance between the head seen by the camera and the screen, even when the head is moving. We also need to know the orientation of the screen, because we can only access the images of the camera. In the beginning, we can make the assumption that the screen is right in front of the camera, "parallel to the face of the person".

To have this, we can use many methods. We tried to implement two of them :

The first on is to use the openCV calibration function, which works well but takes a printed pattern to calibrate, thus the system is not really autonomous. Using this method, we can calibrate the system with a printed pattern, and then use the calibration to compute the position of the gaze on the screen. OpenCV once calibrated gives us the position of the items we "show" it, thus we can have a good 3d position approximation. One of the advantages of this method is also that it has a better understanding of the distortion of the camera, i.e. the fact that the image is not a perfect projection of the 3d world, and that the projection of the plannar image of the camera is not a plannar image in the 3d world.

The second idea is to bypass the need to know exactly where the plane of the screen is by using our own calibration method. The idea is to use a standard calibration for this type of work which is asking the user to look at something we know on the screen, then move. Here, the idea is to ask the user to look at the center of the screen, and assume that the distance face / screen is proportional to a function of the size of the face.

To compute this, we ask the user to fix a point, then move forward and backward. we compute the vectors of the eye gaze, and consider they meet (or closely meet) at a point on the screen. This is not perfect as the gaze is not always perpendicular to the screen, but it can give a good approximation. By computing using smoothed data on this, we can calibrate this function on our own, and thus later use it to determine really quickly the position of the gaze on the screen.

3.4 Estimation of the eye gaze

Once we have the position of the screen in respect to the face of the user, and the estimated vectors of the eye gaze, we must compute the position of the gaze onto the screen.

For this, we introduced two methods : - the first one intersects the two vectors with the plane, then takes the average position of the two points of intersection. We thought this was the most accurate one, and also allowed us to debug more easily. - the seconde one, seen in some other work, just adds the vectors and intersects the result with the plane. We thought this was less accurate, and also more

difficult to debug because we lost information earlier in the process.

3.5 Mouse control and communication

The last part of this project was to use the computed data to communicate with the computer. The first and most obvious way was to use the mouse cursor. We used one/both eye blinking to mimic the clicks of the mouse. Since blinking obscures the eye, we had to use a threshold to detect the blinking, and we also had to use a buffer to smooth the data and avoid false positives, using the last positions computed to decide where the user "clicks".

The second communicated method we used was the morse code. We used the same idea : single eye blink to send a dot, two eye blinks to send a dash, and after some time, it validated the letter. We then used a morse decoder to decode the message and send it to the computer. This worked relatively well since our system was fast and reliable, but we had to use good threshold and buffers to distinguish the "accidental" and natural blinking from intentional communicative one.

3.6 Choice of language and libraries

When we started, we wanted to prototype the system in python and then re implement the needed methods in c++, so we could have a faster thus most precise algorithm. However, we quickly backed away from c++ for two reason :

— first and foremost, we tought the speed of python was too limited but this revealed not true. The used libraries worked really quickly and the system was still fast enough.

— the libraries we wanted to use were not easily available in c++, or required more complications(use bazel for compilation instead of cmake for example), and we tought this would take too much time on implementation, diverting us from spending more time on the main goal and interesting part of the project.

We still tried to use the mediapipe library in c++, but its use was way more complicated than with python, and for our use case, it didn't really make a big time difference, since the core of the python library is fast and not in python.

4 RESULTS

4.1 Experimental protocol

We evaluated our system on short recordings of one user sitting in front of a laptop screen, at a distance of about 60–70 cm. The user was asked to look successively at several targets displayed on the screen (corners, center and some intermediate positions). For each target we recorded the estimated gaze position and compared it to the ground truth position in screen coordinates. We then measured the distance error directly on the physical screen, expressed in centimeters, and we also observed the distribution of this error over all frames.

4.2 Quantitative performances

On these experiments, our system reaches an average gaze position error of approximately 5 cm on the screen at a typical working distance. This precision is not enough to select individual pixels, but it is clearly sufficient to distinguish several large regions of the screen, which matches the use case described in the introduction where the user activates big buttons or zones. Latency is very good : the whole pipeline runs in real time on a standard laptop, and we do not perceive any visible delay between eye movements and the movement of the estimated point on the screen. The detection rate is high as long as the face and eyes are visible ; we mainly lose detection during blinks or very fast head movements, but tracking recovers almost instantly after that.

4.3 Qualitative analysis

Qualitatively, the estimated gaze point follows well the movements of the eyes and stays stable when the user fixes a point on the screen. The system works particularly well in normal lighting conditions, with moderate head motion and without strong occlusions (for example no big reflections on glasses or objects in front of the face). We observe failure cases when the head is too rotated, when the user is too close or too far from the camera, or when the eyes are strongly occluded. Another downside is that the calibration step can take some time and is a bit complicated for non-technical users, since they must follow a specific procedure before using the system. However, once this calibration is done, the interaction with big on-screen buttons or regions feels natural, and the overall reactivity of the

system is satisfying for our target applications.

5 CONCLUSION

In this project we implemented a complete pipeline to estimate the gaze position on a computer screen using only a standard webcam. We built on top of MediaPipe Face Mesh to obtain robust landmarks around the eyes, then used geometric computations and a calibration step to recover the position and orientation of the screen with respect to the face and to intersect the gaze vectors with this plane. Our experiments show that we reach an average error of about 5 cm on the screen at a normal working distance, which is not precise enough for fine pointing but is sufficient for interaction with big buttons or large regions, as described in the introduction. The system runs in real time with a very good latency, so the interaction feels responsive for the user once the calibration is done.

This work also has some limitations. The calibration procedure can be relatively long and a bit complicated for non-expert users, and the robustness of the method degrades when the head is too rotated, the distance to the camera is not in the expected range, or when the eyes are partially occluded. In future work, we could try to simplify the calibration by using more automatic procedures, or by combining our geometric approach with learning-based methods that directly predict gaze from the landmarks. Another interesting direction would be to improve robustness to head pose and lighting changes, so that the system could be used in more realistic, everyday situations.

RÉFÉRENCES

[1] B. Raynal. *Reconnaissance de la pupille par morphologie mathématique.* Actes de l'AFIG, 2007.