Real-Time Gaze Depth Estimation Using a Wearable Eye-Tracking Device

by

Luca Secchieri

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

ETSEIB

UPC

Master's Degree

in

Control System Engineering

Supervisor: Ruggero Carli
Advisor: Manel Frigola Bourlon

Università degli Studi di Padova
Padua, Italy

Abstract

The accurate estimation of gaze depth is crucial for enhancing the functionality of eye-tracking systems, particularly in applications requiring precise three-dimensional gaze localization. This thesis presents a comprehensive approach to real-time gaze depth estimation using Tobii Pro Glasses 3, leveraging machine learning algorithms and precise pupil detection methods. The study begins with a theoretical overview of eye vergence and pupil detection techniques, followed by the development of a regression model trained to estimate gaze depth from the pupil positions. The experimental setup includes real-time video decoding and the analysis of pupil positions to validate the model's performance.

This research contributes to the field of gaze tracking by providing a robust framework for real-time depth estimation, highlighting the importance of precise pupil detection and positional calibration. Future work will focus on refining the model for broader applications and exploring additional features to enhance the accuracy and applicability of gaze depth estimation systems.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Thesis Overview

In recent years, gaze estimation has emerged as a crucial technology in various fields such as human-computer interaction, virtual reality, and assistive technologies. The ability to accurately determine where a person is looking has numerous applications, ranging from enhancing user interfaces to enabling more natural interactions in virtual environments [10]. Traditional gaze estimation techniques typically focus on determining the point of gaze on a two-dimensional screen. However, as 3D technologies and virtual environments become more prevalent, there is a growing need to extend gaze estimation into three dimensions. One key aspect of 3D gaze estimation is determining the depth at which a person is looking, which involves measuring the vergence of the eyes. Vergence refers to the simultaneous movement of both eyes in opposite directions to obtain or maintain single binocular vision. Accurate vergence measurement can provide valuable information about the depth of the gaze, which is essential for applications such as immersive virtual reality systems, advanced driver assistance systems, and robotic control interfaces.

The primary objective of this thesis is to develop a method for estimating gaze depth using images captured by Tobii Pro Glasses 3 ©. These glasses are equipped with high-resolution cameras that can provide detailed pictures of the eyes, enabling precise measurement of pupils' position. From these, and by leveraging the vergence angle, the objective is to derive the depth of gaze from these measurements, thereby extending the capabilities of gaze tracking into three dimensions.

This project is part of a larger initiative focused on developing a robotized wheelchair for individuals with disabilities. The research on gaze depth estimation can be applied to control the wheelchair using only the user's eye movements, captured by the Tobii

Pro Glasses 3. This application has the potential to significantly enhance the mobility and independence of individuals with severe physical impairments, enabling them to navigate their environment through intuitive and natural gaze-based interactions.

1.1.1 Methodology

This section outlines the methodology employed to estimate gaze depth. The process involves several key steps: capturing images, detecting pupil positions, and using these positions to train a regression model for gaze depth estimation. Tobii Pro Glasses 3 are utilized to capture high-resolution images of the eyes. These glasses are equipped with high-accuracy cameras that can record detailed eye movements in naturalistic settings, making them ideal for vergence measurement. The images obtained from the glasses serve as the primary data source for the subsequent analysis. A robust pupil detection algorithm has been developed to accurately locate the pupils in the captured images. This algorithm processes each image to identify the exact positions of the pupils. The detection process involves several steps, including image preprocessing, edge detection, and contour analysis to ensure precise localization of the pupils. The detected pupil positions are then used to train a regression model. This model transforms the detected pupil positions into an estimated gaze depth. Various regression techniques were explored, and the model was trained on a comprehensive dataset to ensure its accuracy and generalizability. The training process involved splitting the dataset into training and validation sets, optimizing the model parameters, and evaluating its performance using standard metrics such as mean squared error.

The implemented methodology was tested in controlled experiments to evaluate its performance. The primary objective was to improve upon the built-in depth estimation provided by the Tobii Pro Glasses 3. The estimated gaze depths from the proposed method were compared against ground truth measurements. The results demonstrated that the proposed approach was more precise and reliable than the built-in estimation, validating the effectiveness of the pupil detection algorithm and the regression model.

Eye Tracking Hardware

For this project, we utilized the Tobii Pro Glasses 3, a state-of-the-art wearable eye tracker designed for real-world data collection. The key features are presented here [38]:

- **Scene camera:** The built-in scene camera boasts a 106° field of view, capturing a comprehensive perspective of the surrounding environment. This feature is crucial for contextualizing the eye tracking data
- **Eye tracking cameras:** The glasses integrate four miniature eye cameras, into scratch-resistant lenses, ensuring high-performance eye tracking without obstructing the wearer's field of view. These cameras provide a video with a resolution of 1024x256 px and a framerate of 50fps.
- **Infrared illuminators:** A total of sixteen infrared lights illuminate the eyes to assist the eye tracking sensors.
- **Eye tracking technique:** From corneal reflection, dark pupil detection, and stereo geometry, a patented 3D eye model is obtained, ensuring accurate gaze data with a sampling frequency of 100 Hz and high precision. The samples contain complete eye tracking information, including 2D gaze, 3D gaze, gaze origin, gaze direction, and pupil diameter.
- **Inertial Measurement Unit:** Tobii Pro Glasses 3 has an IMU sensor that can measure acceleration, rotation, and magnetic field. The accelerometer measures the acceleration along three axes in the head unit coordinate system. The gyroscope measures rotation around three axes (XYZ) at a frequency of 100Hz, while the magnetometer measures the strength of the surrounding magnetic field with a frequency of 10Hz.
- **Microphone:** An integrated microphone records environmental sounds, providing additional context for behavioral analysis

Fig. 1.1 Frontal view of the Tobii Pro Glasses 3

1.2 Structure of the Thesis

The remainder of this thesis is structured as follows: Chapter 2 provides a comprehensive theoretical overview of vergence in human eyes, including the physiological basis and significance of vergence movements. This chapter also explores various pupil detection algorithms, examining their principles, implementations, and effectiveness. Chapter 3 focuses on the development of the machine learning regression model used for gaze depth estimation. It covers the selection of features, model training, and validation processes, along with a discussion of the performance metrics used to evaluate the model. Chapter 4 details the real-time video decoding process, describing the techniques employed to achieve efficient and accurate decoding. This chapter also presents the experimental setup used to test the gaze depth estimation method, and it includes an analysis of the results. Additionally, the estimation of the roll angle of the glasses is discussed, highlighting the methods used for its calculation and its possible use. Finally, Chapter 5 concludes the thesis by summarizing the key contributions and findings, offering a reflection on the implications of the work, and providing suggestions for future research directions to further advance the field.

Chapter 2

Gaze Depth Estimation Using Eye Vergence

2.1 The Human Eyes and 3D Vision

The human eyes are nested in two bony cavities called orbits. Using six extraocular muscles, the eyes perform various rotational movements. The orbits also contain connective tissues, smooth muscles, nerves, blood vessels, and other visual structures [26]. Investigations into the anthropometry of the human eye have found that the radius of the eye globe is approximately 24 mm, with no significant difference in globe dimensions across a wide age range [3]. The component responsible for sharp central vision, known as foveal vision, is the retinal fovea. This small central pit in the macula lutea of the retina is where the visual axis passes through the center of the fovea and the center of the pupil. For clear acute vision, the image of the object must be held steadily on the central, foveal region of the retina, which imposes a two-fold limitation problem: maintaining the steadiness of the object and ensuring the correct distance from the fovea.

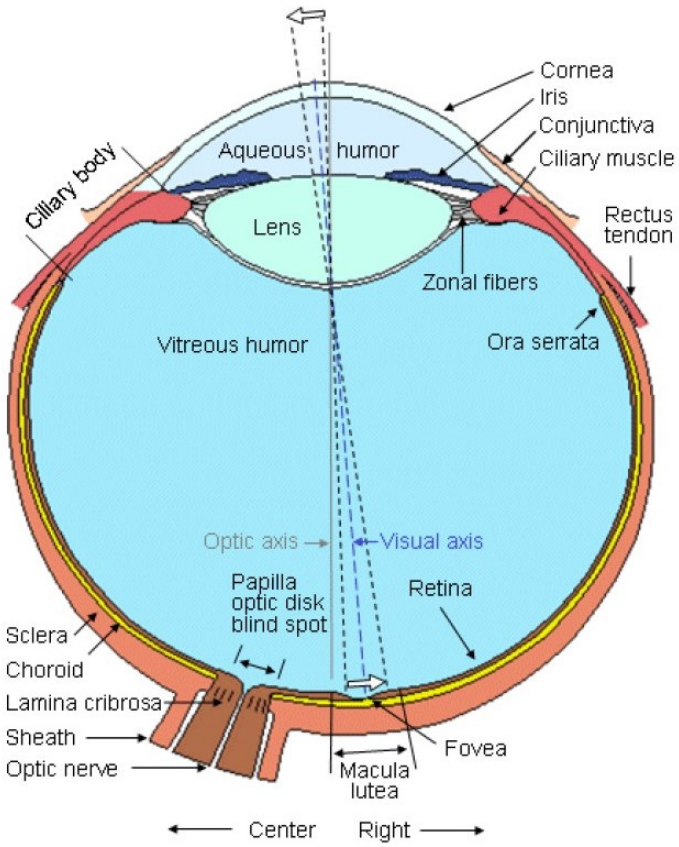


Fig. 2.1 Horizontal section of the right human eye seen from above [22].

In video-based eye tracking, 3D gaze estimation aims to determine the gaze vector in the real world based on the estimated center of the pupil within the captured eye image. This process necessitates the extraction of depth information from the visual data [24].

The human brain uses several sources of information for 3D vision and reconstruction, including both monocular and binocular cues. Monocular cues, such as occlusion, provide depth information from a single eye, while binocular disparity, which arises from the slight differences in the images seen by each eye, plays a critical role in depth perception [18]. Additionally, the brain integrates input from three oculomotor systems to perceive depth: accommodation, miosis, and vergence [28].

Depth perception relies on these cues to create a coherent sense of space and distance. Accommodation is the process by which the vertebrate eye adjusts its optical power to maintain a clear image or focus on an object as its distance from the eye varies [7]. This adjustment involves changing the shape of the lens within the eye,

which is controlled by the ciliary muscles. Accommodation helps provide a continuous and clear image of objects at different distances.

Miosis refers to the constriction of the pupil, which adjusts the amount of light entering the eye [23]. This process is primarily influenced by the brightness of the environment, but it also contributes to depth perception by improving the depth of field—the range of distances within which objects appear in focus. Smaller pupils increase the depth of field, allowing the eye to better perceive depth.

Vergence involves the simultaneous movement of both eyes in opposite directions to maintain single binocular vision. To focus on nearby objects, our eyes turn inward in a coordinated movement called convergence. Conversely, when shifting focus to distant objects, the eyes rotate outward in a process known as divergence. Therefore, when focusing on an object binocularly, the eye vergence angle (EVA) is the angle between the visual axes of both eyes [14]. Generally, a closer object requires a larger EVA, indicating the eyes are turning more inwards to converge. Conversely, focusing on distant objects results in a smaller EVA [2]. As the distance of the object becomes significantly larger than the space between the pupils (interpupillary distance), the EVA approaches zero. At this point, the lines of sight for both eyes are virtually parallel (see Fig.2.2).

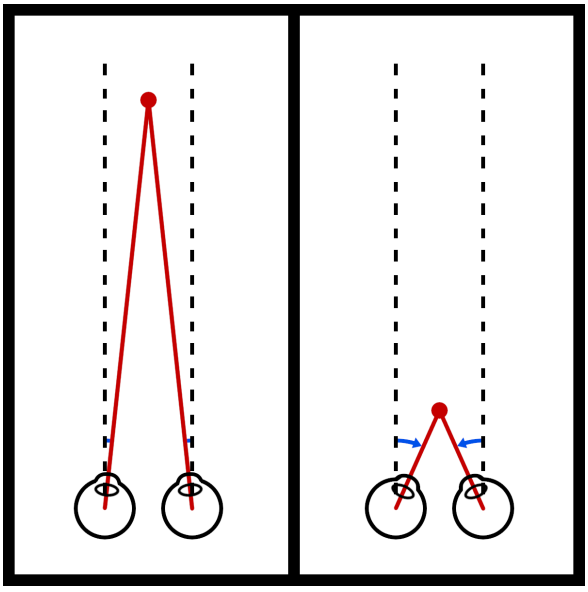


Fig. 2.2 The two-eye vergence depth cue. The vergence angle varies with the distance of the object from the eyes. Public Domain Image, source: Christopher S. Baird

By integrating these mechanisms, the human visual system can effectively interpret and construct a three-dimensional representation of the surrounding environment. The

interplay between accommodation, miosis, and vergence allows for a dynamic and adaptable perception of depth, essential for tasks ranging from basic navigation to complex manipulations in three-dimensional space. Understanding these processes is fundamental for developing advanced gaze estimation technologies, which can leverage the principles of human vision to enhance artificial systems in fields such as robotics and assistive technologies.

Estimating the 3D position of the gaze fixation point is a complex task that begins with detecting the pupils' positions. The primary challenge arises from the fact that the pupil rotates on a sphere, representing an approximation of the eyeball. The exact center of this sphere is difficult to define and varies significantly depending on the positioning of the glasses and the individual user's anatomy. Additionally, even if we accurately determine the eyeball's center, the center of the eye's rotation and the visual axis typically do not align [13], further complicating the estimation process. This misalignment adds another layer of difficulty to achieving precise 3D gaze estimation.

For the design of a 3D eye-tracking system, measuring accommodation requires perfectly controlled conditions and complex equipment, such as ultrasound biomicroscopy. This approach is not suitable for prolonged continuous use. Miosis, while easily measurable, is highly sensitive to ambient light conditions. Therefore, among the three sources of information from the oculomotor system, vergence is the most practical for estimating the gaze position or Point of Regard (PoR). In video-based 3D eye-tracking systems, vergence is the only signal that is both robust and straightforward to measure [24].

Existing techniques for 3D gaze estimation fall into two primary categories: methods based on geometrical models of the eye, and methods based on interpolation, which directly map eye position to PoR. Interpolation methods often include regression models to predict PoR from eye position data, utilizing both linear and non-linear regression techniques to accurately estimate gaze direction.

Observation Distance and Vergence Angle

To gain a deeper understanding of the problem, we first need to calculate the theoretical vergence angle as a function of distance. The key parameters in this calculation include the positions of the eyes and the point of fixation in front of them. A ray is traced from each eye towards this point, and for our theoretical framework, we assume that the point of fixation is directly ahead, positioned centrally between both eyes. To analyze the relationship between eye vergence and distance, we define z as the object distance,

θ as the vergence angle, and d as the interpupillary distance (IPD). The vergence angle θ can be calculated using the following formula:

$$\theta = 2arctan\left(\frac{d}{2z}\right) \tag{2.1}$$

Given the symmetrical positioning of the eyes and the fixation point directly ahead, the vergence angle can be derived using basic trigonometry. The eyes and the fixation point form an isosceles triangle where the base is d and the height is z. The vergence angle is the angle at which the lines of sight from the eyes intersect at the fixation point. To visualize the relationship between, we can plot θ as a function of z using the previous formula. We'll use the population mean interpupillary distance (IPD) of 63 mm [9] for our calculations.

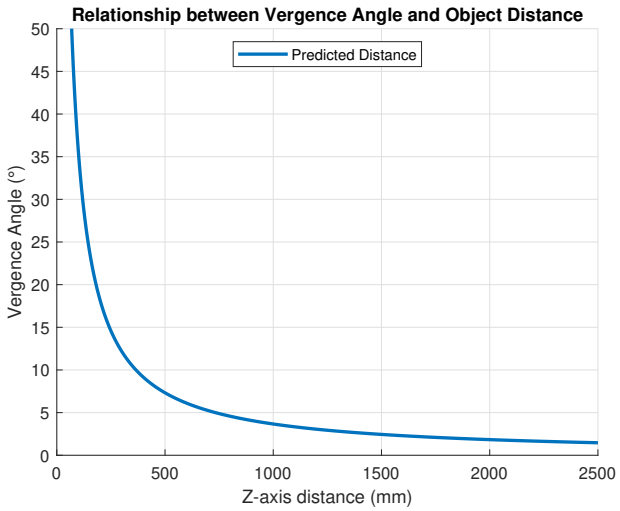


Fig. 2.3 Exponential Relationship Between Vergence Angle and Object Distance. This figure shows how the vergence angle of the eyes increases exponentially as the object distance decreases.

Moreover, because the vergence angle is determined by the distance to the eyes, considering any point on a hypothetical sphere centered around the eyes would yield similar results. This is due to the symmetrical nature of the sphere where every point at the same distance from the eyes would require the same vergence angle for fixation. This theoretical understanding is crucial for accurately modeling and predicting vergence behavior in various applications, such as in the design of eye-tracking systems and in studying visual perception dynamics.

Interpupillary Distance

Another key factor in human binocular vision that varies with object depth is interpupillary distance (IPD). This refers to the distance measured between the centers of the pupils in each eye [9]. Changes in the depth of a verged object can significantly impact the interpupillary distance [16]. Specifically, when the eyes focus on a nearby object, the IPD value decreases, while focusing on a distant object results in a larger IPD [2]. There is substantial evidence indicating that infrared eye-tracking technology is sufficiently reliable for detecting these small changes in IPD [19].

2.2 Problem Statement

In modern assistive technologies, particularly those aimed at enhancing the mobility of individuals with disabilities, accurate and reliable eye-tracking systems are essential. These systems enable hands-free control of devices, such as robotized wheelchairs, through eye movements. One critical component of such systems is the ability to accurately estimate gaze depth, which is derived from the vergence of the eyes. However, current eye-tracking solutions, such as those provided by the Tobii Pro Glasses 3, often fall short in precision when estimating gaze depth, especially in dynamic and uncontrolled environments.

The challenge lies in extracting accurate gaze depth from the images captured by the Tobii Pro Glasses 3. This process involves leveraging the vergence angle and interpupillary distance (IPD) to make precise depth estimations. The initial step in this approach is to develop a robust procedure for detecting pupils in the eye images. Accurate pupil detection is fundamental as it directly influences the subsequent steps of extracting the IPD.

Once the pupil positions are accurately detected, the next step involves training a regression model. This model is designed to transform the detected pupil positions into accurate predictions of gaze depth. The effectiveness of this model depends on its ability to learn from the training data and generalize to new, unseen data. Given the variability in eye movements and the dynamic nature of real-world environments, this presents a significant challenge.

Additionally, the discrepancy between accommodation and vergence in stereo display systems can lead to inaccuracies in depth perception, contributing to user discomfort and reduced precision in gaze estimation. While infrared eye-tracking technology shows potential for measuring small changes in IPD and vergence, there is still a need for a more robust and practical solution for real-time depth estimation.

Therefore, this thesis aims to address these challenges by developing a method to accurately estimate gaze depth using the vergence information obtained from Tobii Pro Glasses 3. By improving the built-in depth estimation capabilities, this research seeks to enhance the reliability and precision of eye-controlled assistive technologies, thereby improving the quality of life for individuals with disabilities through more effective and user-friendly robotized wheelchairs.

2.3 Pupil Detection

In the process of estimating gaze depth from eye images, the first crucial step is accurately detecting the pupils. The glasses capture four frames containing the eyes, each illuminated to highlight the pupils for more precise detection. The accuracy of pupil detection is essential, as it directly impacts the subsequent steps in the estimation of gaze depth. Below is a sample image obtained with the Tobii Pro Glasses 3, showing the four frames containing the eyes. This image serves as the input for the pupil detection process, which identifies and tracks the position of the pupils across the frames.

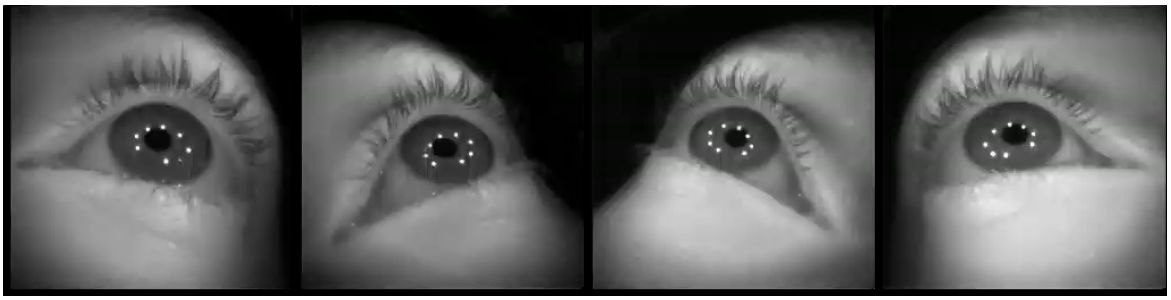


Fig. 2.4 Example of a frame obtained by the four internal cameras mounted on the Tobii Pro 3 Glasses.

2.3.1 First Experimental Setup

The experimental setup involved positioning a target, consisting of a small card with a distinctive bullseye pattern, in front of a long table spanning a length of about 4 meters. The primary objective was to capture videos of the target at varying distances, ranging from 50cm to 350cm, with intervals of 25cm between each distance. This provided a dataset of 13 videos in total. To ensure precision in distance measurements, a tape measure was securely positioned to the table. This allowed for accurate determination of the distances between the target and the observer.

In the first experiment, all targets were aligned directly in front of the observer's eyes. While efforts were made to maintain this alignment consistently across all distance settings, it's important to note that a high degree of precision in alignment was not strictly necessary. The key focus of the experiment was on capturing the variation of the distance between the observer's pupils as the target was moved further away, rather than achieving absolute precision in alignment. Therefore, there was a certain level of tolerance allowed regarding the alignment of the targets, as long as the general positioning in front of the observer's eyes was maintained. This approach ensured that the experiment could proceed smoothly without overly stringent requirements for alignment precision. In the same way, the position of the observer's head was fixed throughout the experiment. This ensured that the perspective from which the videos were captured remained constant, and so facilitating analysis and comparison across different distance settings.
Each video clip was recorded for 10 seconds, striking a balance between collecting sufficient data and maintaining the observer's focus on the target. Particular attention was paid to ensuring that the gaze was constantly focused on the target, as any distraction would have led to inconsistencies in the following analysis.

Pupil Detection Procedures

Three distinct procedures were designed to analyze the extracted frames and identify the precise coordinates (x, y) of the pupils in each image. The goal was to develop robust and accurate methods capable of reliably detecting pupil positions across a variety of conditions.

2.3.2 Pupil Detection Using Binarization

The first approach for pupil detection requires the binarization of the source image and for this reason, it will be called the binarization method. Binarization is a fundamental image processing operation that converts a grayscale image into a binary image, where each pixel is either fully black (0) or fully white (255). This binarization method was implemented in Python [39] using the open-source computer vision library OpenCV [1]. The steps of this first procedure are presented below:

1. **Pre-processing of the image:** In order to reduce the possibilities of false detections, it's necessary to prepare the image. Firstly a median blur filter is applied to reduce the size of the reflections of the infrared lights (see figure 3.8).

After that, the erosion function is applied to further suppress reflections: it removes brighter pixels (potential reflections) on the blurred image, effectively emphasizing darker, underlying features. Morphological erosion removes floating pixels and thin lines so that only substantive objects remain. Remaining lines appear thinner and shapes appear smaller [36].

2. **Binarization:** The purpose of binarization in this context is to segment the image into regions of interest based on their intensity values, in particular, the goal is to separate the pupil from the rest of the eye. After some trial and error tuning, the best results were obtained by applying a threshold value of 220: pixels with intensity values greater than or equal to this threshold are considered part of the foreground (pupil) and are assigned a value of 255, while pixels with intensity values below this threshold are considered part of the background and are assigned a value of 0 (see Fig 2.6a).

3. **Contours extraction:** Contours extraction is a process used to detect and represent the boundaries of objects in an image. In this case, the cv2.findContours() function is used to extract contours from the binary image (see figure 2.6b).

4. **Enclosing circles evaluation:** After extracting all the contours detected in the image, the code iterates through each of them. For each contour, the minimum enclosing circle is computed using cv2.minEnclosingCircle(). This function provides a good estimation of the pupil since it is a fairly circular object.

   Now, as the structure of the image was known beforehand, we could take advantage of this information and estimate that the pupil is positioned close to the middle point of the image. In order to favor those contours that are closer to the above-said point, the distances between the center of the enclosing circles and a predefined center are computed. Additionally, to avoid false detections, only circles with a radius in the range between 8 and 20 are considered. Finally for every iteration, if the contour satisfies these conditions and is closer to the center than any previously encountered contour, the final_cnt variable is updated to store this contour.

5. **Pupil coordinates:** From the best contour selected, the center and radius of the enclosing circle are saved, and a variable indicating that the pupil extraction was successful is set to true. Then two circles are drawn on the original image: one representing the pupil with a red border and the other representing its center with a green border (see Fig 2.12).

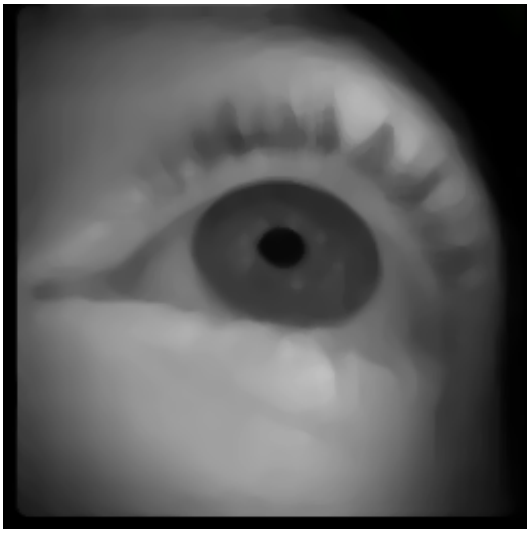
If the attempt to find the enclosing circle fails (e.g., due to an empty contour), an exception is thrown and a message indicating that the pupil was not detected is printed, so that the frame can be discarded.

```
def detect_pupil (self):
    #Pre-processing
    blurred_img = cv2.medianBlur(self._img, 5)
    gray_img = cv2.cvtColor(blurred_img, cv2.COLOR_BGR2GRAY)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,
       3))
    eroded_img = cv2.erode(gray_img,kernel,iterations = 1)

    #Binarization
    ret,thresholded_img = cv2.threshold(eroded_img,220,255,cv2.
       THRESH_BINARY_INV)

    #Contours extraction
    contours, hierarchy = cv2.findContours(thresholded_img, cv2
       .RETR_LIST, cv2.CHAIN_APPROX_NONE)

    #Select the best contour (closest to center)
    flag = 5000
    final_cnt = None
    for contorno in contours:
        (x,y),radius = cv2.minEnclosingCircle(contorno)
        distance = abs(self._center[0]-x)+abs(self._center[1]-y
           )
        #Update the best contour
        if distance < flag and radius<20 and radius>8:
            final_cnt = contorno
            flag = distance

    try:
        (x,y),radius = cv2.minEnclosingCircle(final_cnt)
        pupil_center = (int(x),int(y))
        radius = int(radius)

        #Draw the circles representing the pupil and its center
        cv2.circle(self._img,center,radius,(0,0,255),2)
```

```
32          cv2.circle(self._img,center,2,(0,255,0),2)
33
34          #Saving the pupil center coordinates
35          self._pupil = (pupil_center[0],pupil_center[1],radius)
36          self.detected = True
37
38      except cv2.error as e:
39          print('Pupil not detected')
```

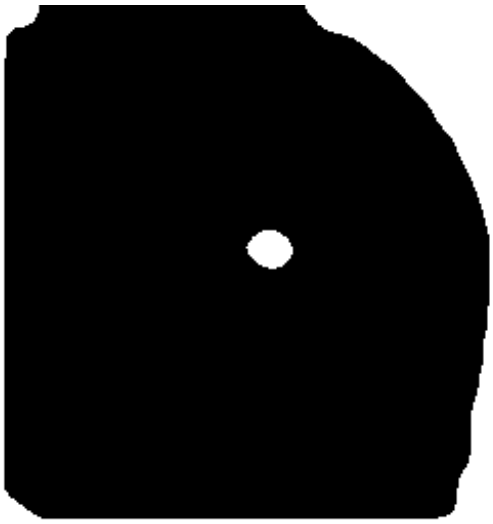
Listing 2.1 Python code for the binary method pupil extraction

(a) The binarization method works on one image at a time

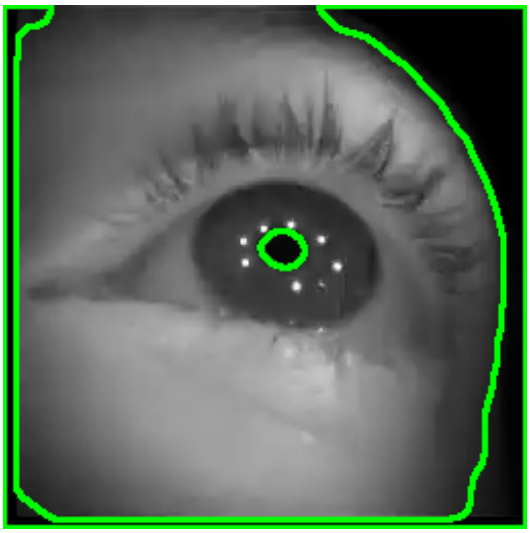


(b) Filtering of the image using median blur

Fig. 2.5 Intermediate steps of the pupil detection process



(a) Output of the binarization using a fixed threshold



(b) The contours found are highlighted In green

Fig. 2.6 Intermediate steps of the pupil detection process

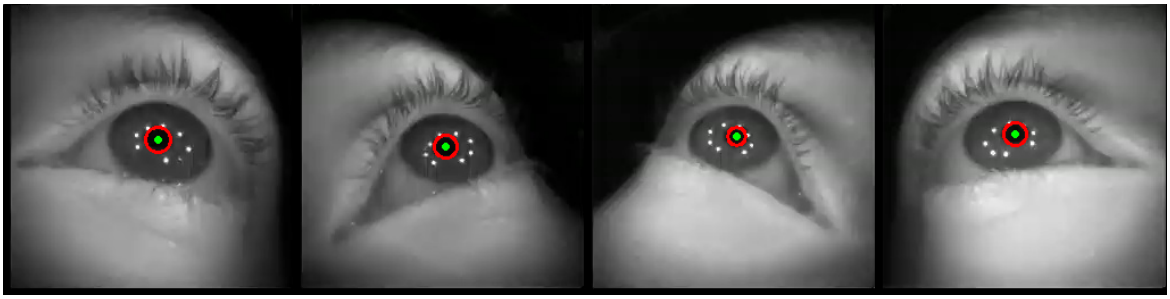


Fig. 2.7 Final result: the detected pupil (red) and its center (green)

Results

Using Python and OpenCV, we processed the video dataset. On average, 57 frames were extracted from each of the 13 clips, resulting in a total of 744 images. The binarization method effectively filtered out frames where eyes were entirely closed due to blinking. However, limitations emerged in handling partially closed eyelids and corrupted frames (see figure 2.8), leading to occasional false detections. After the manual removal of these false positives from the dataset, only 590 frames remained. From this point onwards, just these clean frames were used to test the pupil detection processes. This strategy successfully extracted pupil coordinates from each of these refined frames, resulting in a total of 2,360 (4 pairs per image * 590 images) data

points stored in a table. It's worth noting that this technique can achieve a single-pixel precision in detecting the pupil center.

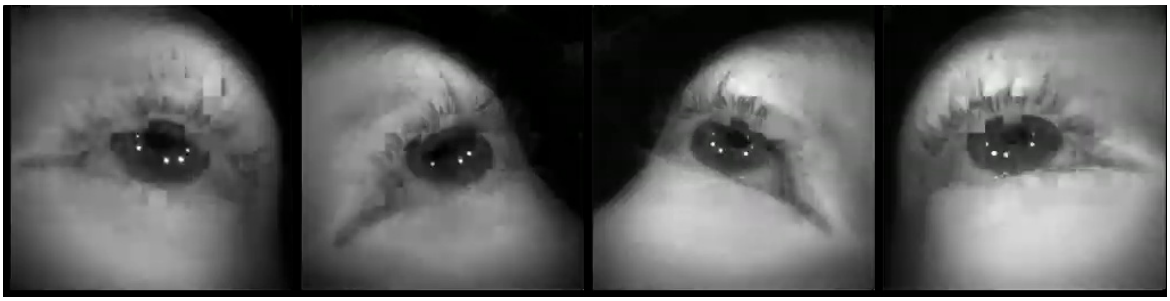


Fig. 2.8 Example of corrupted frame

2.3.3 Pupil Detection Using Hough Transform

The second approach for pupil detection involves the Hough transform to localize circles in the image. Introduced in 1962 by Paul Hough, the primary motivation behind its invention was to provide a robust and efficient means of detecting lines in noisy or cluttered images without relying on explicit segmentation or thresholding techniques. Let (x_i, y_i) denote a point in the xy-plane and consider the general equation of a straight line in slope-intercept form: $y_i = ax_i + b$. Infinitely many lines pass through (x_i, y_i), but they all satisfy the equation $y_i = ax_i + b$ for varying values of a and b. However, writing this equation as $b = -x_i a + y_i$ and considering the ab-plane (also called *parameter space*) yields the equation of a single line for a fixed point (x_i, y_i). Furthermore, a second point (x_j, y_j) also has a single line in parameter space associated with it, which intersects the line associated with (x_i, y_i) at some point (a', b') in parameter space, where a' is the slope and b' the intercept of the line containing both (x_i, y_i) and (x_j, y_j) in the xy-plane (we are assuming, of course, that the lines are not parallel). In fact, all points on this line have lines in parameter space that intersect at (a', b'). In principle, the parameter space lines corresponding to all points (x_k, y_k) in the xy-plane could be plotted, and the principal lines in that plane could be found by identifying points in parameter space where large numbers of parameter-space lines intersect. However, a difficulty with this approach is that a, (the slope of a line) approaches infinity as the line approaches the vertical direction. One way around this difficulty is to use the normal representation of a line: $xcos(\theta) + ysin(\theta) = \rho$. The computational attractiveness of the Hough transform arises from subdividing the $\rho\theta$ parameter space into so-called *accumulator cells* [11]. Each cell in the accumulator corresponds to a specific slope and intercept pair. As the algorithm scans through the image, it accumulates votes in

the corresponding cells of the accumulator for every edge point that could potentially lie on a line defined by those parameters. By accumulating votes in the accumulator cells, the algorithm identifies peaks in the accumulator space, indicating the presence of shapes in the image.

Although the focus thus far has been on straight lines, the Hough transform is applicable to any function of the form $g(v, c) = 0$, where $\mathbf{v}$ is a vector of coordinates and c is a vector of coefficients. For example, points lying on the circle

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2 \tag{2.2}$$

can be detected by using the basic approach just discussed [11]. By analyzing the accumulator array generated from this transformation, the algorithm can efficiently identify circular patterns present in the image, which in this context, correspond to pupils.

The steps of this second procedure, also implemented in Python with the OpenCV library, are presented here.

1. **Pre-processing of the image:** As previously done, median blur filter is applied to the grayscale image. This filter reduces noise in the image while preserving edges, which can improve the accuracy of subsequent edge detection.

2. **Canny Edges Extraction:** The Canny edge detection algorithm is applied using the function cv2.Canny(). This algorithm detects edges in the image by looking for areas of high gradient magnitude (see figure 2.9). The parameters 50 and 100 specify the lower and upper thresholds for edge detection. Pixels with gradient magnitudes below the lower threshold are discarded as non-edges, while those above the upper threshold are considered strong edges. Pixels with gradient magnitudes between the two thresholds are considered weak edges unless they are connected to strong edges.

3. **Detection of Hough Circles:** The Hough Circle Transform algorithm is used to detect circular shapes in the edge-detected image. This method is particularly suitable for detecting circles, even in the presence of noise or incomplete edges. The parameters passed to cv2.HoughCircles() had to be tuned manually, but the final result is heavily influenced by the edges detected in the previous step.

4. **Processing Detected Circles:** If circles are detected in the image, the code proceeds to process the detected circles. For each detected circle, a green dot is

drawn at its center and a red circle is drawn around it with the detected radius. These, as in the binary method, serve as visual indicators of the detected pupil. It's important to underline that this procedure shows all the circles that are detected in the image, and for this reason it was crucial to tune the parameters to make it sensitive to the pupil only, and so avoiding false positives.

```
def detect_pupil(self):

    #Pre-processing
    gray = cv2.cvtColor(self._img, cv2.COLOR_BGR2GRAY)
    gray = cv2.medianBlur(gray, 7)

    #Canny edges extraction
    edges = cv2.Canny(gray,50,100)

    #Detection of the Hough Circles
    circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1,
       150,
                               param1=100, param2=15,
                               minRadius=5, maxRadius=15)

    if circles is not None:
        circles = np.uint16(np.around(circles)) #This converts
           the coordinates and radii of the detected circles to
            integers.
        for i in circles[0, :]:
            center = (i[0], i[1])
            radius = i[2]

            #Draw the circles representing the pupil and its
               center
            cv2.circle(self._img, center, 2, (0, 0, 255), 2)
            cv2.circle(self._img, center, radius, (0, 255, 0),
               1)

            #Saving the pupil center coordinates
            self._pupil = (center[0],center[1],radius)
            self.detected = True
```

Listing 2.2 Python code for the Hough method pupil extraction

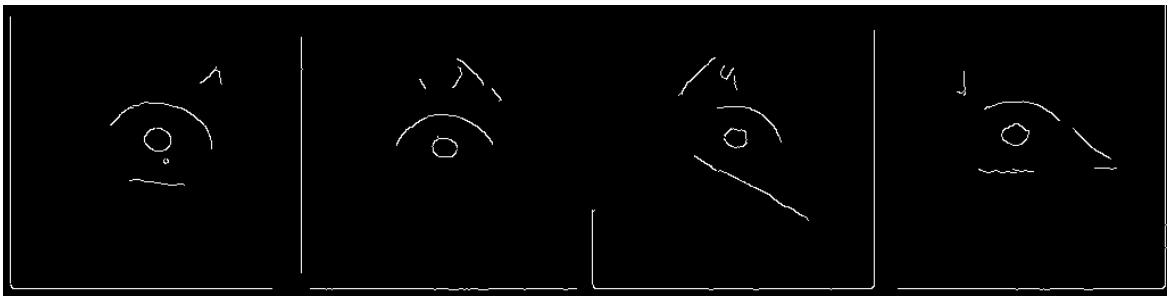


Fig. 2.9 Output of the Canny edge detector

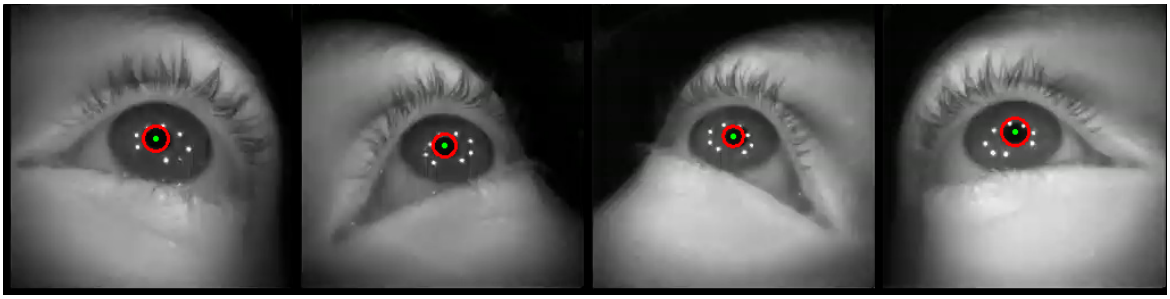


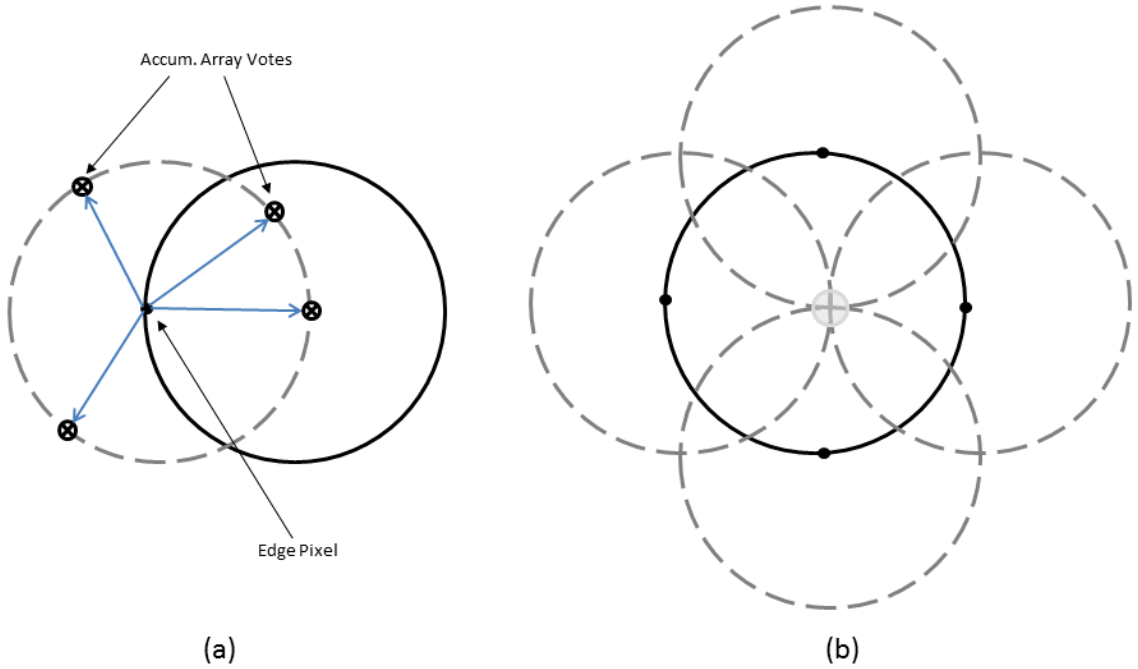
Fig. 2.10 Final result: the detected pupil (red) and its center (green)

Results

While this method successfully identified the presence of pupils in the frames, it encountered limitations in precisely determining the center coordinates of these pupils. These coordinates exhibited inconsistencies between consecutive frames, even though minimal to no movement was expected. This unexpected behavior could be attributed to small variations between frames in the edges detected by the Canny edge detection algorithm. These variations in edge detection can significantly impact the estimated center of the pupil. The Circular Hough Transform (CHT), which relies on edge points to identify circles, can be misled by these slight edge discrepancies. As a result, the estimated center of the pupil, based on the CHT output, can shift by several pixels between consecutive frames, even when the actual pupil position remains relatively unchanged. Consequently, the dataset obtained from this second approach had to be discarded due to the inconsistencies and unreliability of the results.

2.3.4 Pupil Detection Using MATLAB Primitives

The third and last method for pupil detection was done using MATLAB. This method revolves around the *imfindcircles* function, which uses a Circular Hough Transform (CHT) based algorithm for finding circles in images. In the CHT process, candidate pixels that have high gradients are identified and permitted to contribute "votes" to an accumulator array. These candidate pixels employ a predetermined voting pattern, often forming a complete circle of fixed radius around them. The votes cast by candidate pixels aligned with an image circle tend to accumulate at the bin in the accumulator array corresponding to the circle's center (see figure 2.11). Consequently, estimating the centers of these circles involves identifying peaks within the accumulator array. The width of the voting interval, between points c_{min} and c_{max} in the figure, is determined by the radius range defined by r_{min} and r_{max} [34].



.

Fig. 2.11 Figure 1a shows an example of a candidate pixel lying on an actual circle (solid circle) and the classical CHT voting pattern (dashed circles) for the candidate pixel. Figure 1b shows an example of the candidate pixels (solid dots) lying on an actual circle (solid circle), and their voting patterns (dashed circles) which coincide at the center of the actual circle [34]

The actual steps of this third approach for pupil detection are presented below.

1. **Pre-processing of the image:** As usual, a Gaussian filter is applied to remove noise from the image. Following that, a disk-shaped structuring element for morphological operations is created. Morphological opening is useful for removing small objects and thin lines from an image while preserving the shape and size of larger objects in the image [36]. In this case, it enhances the pupil and filters infrared light reflections.

2. **Circle detection:** To detect the pupils in the image, the function *imfindcircles* is used. It returns the center coordinates and radii of the detected circles, along with a metric that indicates the strength of the detection. The parameters are tuned to avoid false positive detection, but even in the case of more circles found, the pupil is detected with the most "confidence".

3. **Handling detected circles:** As the function stores the centers of all the detected circles in the image, it is crucial to select the right circle (the one corresponding to the pupil). The function also returns a column vector, called metric, containing the magnitudes of the accumulator array peaks for each circle (in descending order). So when more than one circle is detected, the center coordinates of the pupil are always stored in the first row of the returned array. In any case, the coordinates values were also double-checked manually to avoid mistakes.

```
I = I0(:,1+rows*i:rows*(i+1)); % cut the image to work with 1
   eye at a time

%Pre processing
H1 = fspecial("gaussian",5, 5/3);
IF = imfilter(I,H1);

SE1 = fspecial("disk",5) > 0; % pupil enhancement & Infra-red
   light reflexions filtering
PE = imopen(IF,SE1);

%Pupil detection
[centers,radii,metric] = imfindcircles(PE,[8 16], "
   ObjectPolarity", "dark", "Sensitivity", 0.92);
h = viscircles(centers,radii, 'LineWidth',1); %circles
   visualization

if size(centers, 1) == 1 %If 1 circle is detected
```

```
15      centri(j,1+(i*2)) = centers(1);
16      centri(j,2+(i*2)) = centers(2);
17
18  elseif size(centers, 1) > 1 %If more than 1 circle is detected
19
20      centri(j,1+(i*2)) = centers(1,1);
21      centri(j,2+(i*2)) = centers(1,2);
```

Listing 2.3 Sample MATLAB code

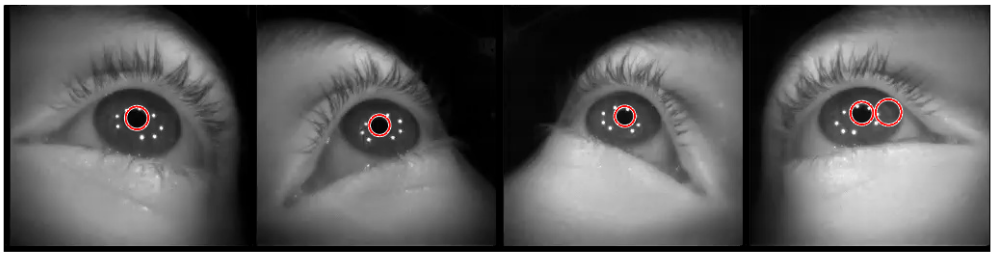


Fig. 2.12 Example of pupil detection using *imfindcircles*. In the last eye 2 circles are found, one centered in (123.14,112.32) and the other (150.74,111.75). As stated before, the correct one will have a higher value in its corresponding metric, which are 0.29 and 0.12.

Results

Since the method aimed to extract pupil coordinates from all four images in a frame, a single undetected pupil led to discarding the entire frame. Consequently, in the matrix containing the pupil center coordinates, all rows where at least one pupil center value was missing (represented by zero) were removed from the matrix.
The process demonstrated impressive strengths. It achieved high precision, which means that it only identified actual pupils and no false positives were present. Furthermore, it exhibited sub-pixel precision, indicating its capability of locating pupil centers with exceptional accuracy. However, these strengths were overshadowed by a high rate of missed detections. This limitation resulted in the removal of a significant portion of the frames. Despite processing an initial set of 590 frames, the method only successfully identified the pupil centers in 315 of them. This substantial loss of data due to missed detections ultimately led to the abandonment of this method.

2.4 More Results on Pupil Detection

Initial testing of the pupil detection procedure involved using a binarization method on images obtained from the Tobii Pro Glasses 3. This approach worked adequately under controlled conditions; however, it required manual adjustment of the threshold value based on varying light conditions and gaze directions, which proved to be tedious and impractical for real-time applications.

The binarization method encountered significant challenges when there was low contrast between the iris and the pupil, such as in individuals with dark eyes. This issue led to unreliable pupil detection.

(a) (b)

Fig. 2.13 Even by manually adjusting the threshold value, static thresholding fails to detect the pupil due to light reflections and low contrast (2.13a). Using adaptive thresholding we are able to correctly detect the pupil in a wider range of conditions (2.13b).Eye images from [21].

To address this problem, static thresholding was replaced by an adaptive thresholding function, which dynamically adjusts the threshold value for different regions of the image, making it more robust to changes in lighting and iris-pupil contrast. This method significantly improved the accuracy and reliability of pupil detection across a wider range of conditions and eye colors (see Fig2.14).

Fig. 2.14 Results of pupil detection using adaptive thresholding on different iris colors. Eye images from [21].

Chapter 3

Regression Model for Depth Estimation

The objective of the following part is to develop a regression model capable of estimating gaze depth using pupil position data. The experiment utilized the datasets containing pupil coordinates and corresponding gaze depths previously collected. MATLAB's (version R2024a) regression learner app was employed to create, train, and evaluate various models.

3.1 Machine Learning Theory

In the field of machine learning, algorithms are designed to mimic human learning by extracting knowledge directly from data. Unlike traditional programming methods that rely on predefined models, machine learning utilizes computational techniques to uncover patterns and relationships within the data itself. This allows the algorithms to continuously improve their performance as the volume of available training data increases. In particular, machine learning uses two types of techniques: supervised learning (such as classification and regression), which trains a model on known input and output data so that it can predict future outputs, and unsupervised learning (such as clustering), which finds hidden patterns or intrinsic structures in input data [35]. Supervised learning requires learning a mapping between a set of input variables X and an output variable Y and applying this mapping to predict the outputs for unseen data. Supervised learning is the most important methodology in machine learning and it also has a central importance in the processing of multimedia data [8].

Regression in Supervised Learning

Regression is a powerful technique within supervised learning that focuses on predicting continuous outputs [25]. Unlike classification, which deals with assigning data points to discrete categories, regression aims to establish a mathematical relationship between input features (X) and a continuous output variable (Y). This relationship can then be used to estimate the value of Y for new, unseen data points. There are various regression algorithms, each with its own strengths and weaknesses.

3.1.1 Introduction to Machine Learning

Starting with some terminology, we can define:

- **Domain set:** This is the set of objects that we may wish to label [32]. It is typically denoted by the letter $\mathcal{X}$ and can be visualized as a multidimensional space where each dimension corresponds to a specific input feature. In our case, $\mathcal{X}$ corresponds to the 8-dimensional space of possible pupil coordinates.
- **Label set:** The label set, also known as the output space, represents the collection of all possible output values that a machine learning model can predict. It is typically denoted by the letter $\mathcal{Y}$ and can be discrete (categorical) or continuous (numerical) depending on the type of learning task (classification or regression). In our case, $\mathcal{Y}$ corresponds to all the possible gaze depth values.
- **Training set:** The training set is a subset of the entire dataset used to train a machine learning model. It is defined as $\mathcal{D} : \{(x_i, y_i), i = 1, ..., m\} x_i \in X, y_i \in Y$. It consists of labeled examples, meaning each data point in the training set includes both the input features ($\mathcal{X}$) and the corresponding desired output values ($\mathcal{Y}$).
- **Learning Algorithm:** The learner is requested to output a prediction rule, $h : \mathcal{X} \to \mathcal{Y}$. This function is also called a predictor or a hypothesis and can be used to predict the label of new domain points. We use the notation $A(S)$ to denote the hypothesis that a learning algorithm, A, returns upon receiving the training sequence S, so $A(S) \to h \in H$, where H is the hypothesis class [32].
- **Error function:** The error function, also known as the loss function, is a mathematical measure that quantifies the difference between the model's predicted outputs and the actual labels in the training data. Given $z = (x, y)$ and the

loss function $l(h, z)$, it measures how good is the function h to describe the data point z, namely $h(x) \simeq y$. The learning algorithm aims to minimize this error function by iteratively adjusting the model's internal parameters during the training process. A common error function for regression problems is the mean squared error (MSE).

True Risk and Empirical Risk

The true risk, also known as generalization error is defined as follows:

$$L_D(h) = \mathbb{E}_D[l(h, z)] \tag{3.1}$$

and it represents the expected performance of a machine learning model on unseen data drawn from the actual underlying distribution that generated the data. Unfortunately, the true risk is often impossible to calculate directly because we typically don't have access to the entire data distribution.
The empirical risk, also known as training error,

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} l(h, z) \tag{3.2}$$

represents the average performance of a machine learning model on the training data itself. Since the training sample is the snapshot of the world that is available to the learner, it makes sense to search for a solution that works well on that data. The goal of the learning algorithm is to compute a predictor h_S that minimizes L_S, and for this reason it is called *Empirical Risk Minimization* or ERM for short [32]. However, the key challenge lies in ensuring this learning generalizes to unseen data – reflected by a low true risk. So we can define the generalization gap as:

$$Generalization\, Gap = L_D(h) - L_S(h) \tag{3.3}$$

In other words, the goal of machine learning is to find a model that minimizes the true risk, based on its empirical risk.

The Bias-Variance Tradeoff

The bias-variance tradeoff is a fundamental concept in machine learning that refers to the tension between a model's ability to fit the training data well and its ability to generalize to new, unseen data.

- **Bias**: refers to the systematic underestimation or overestimation of the true relationship between the input features and the output variable. It arises from simplifying assumptions made by the model or limitations in its capacity to capture the complexity of the data. In other words, bias is inherent to your model. High bias leads to underfitting, where the model fails to capture the underlying patterns in the data, resulting in poor performance on both the training and testing data. This leads to a high empirical risk and likely a high true risk as well.

- **Variance**: represents the model's sensitivity to the specific training data used. A model with high variance memorizes the training data too well, including noise and irrelevant details. This can lead to overfitting, where the model performs very well on the training data but fails to generalize to unseen data. In other words, this results in a low empirical risk but a high true risk.

Train-Validation-Test Split

A more accurate estimation of the true risk can be obtained by using some of the training data as a validation set [32]. In most practical applications, we split the available examples into three sets.

1. **Training Set:** This is the primary dataset used to train the model. The model learns patterns and relationships within the training data by fitting its internal parameters to minimize the prediction error on these examples.

2. **Validation Set:** This second dataset serves a crucial role in model selection. After training various models on the training set, we evaluate their performance on the validation set. The model that generalizes best to the unseen validation data is chosen as the final model.

3. **Test Set:** Finally, we have the test set, often referred to as the "holdout" data set if it's completely untouched during training. This set provides an unbiased estimate of the final model's true error on unseen real-world data. The performance on the test set reflects how well the model generalizes and performs in practical scenarios.

By splitting the data and following this approach, we try to avoid overfitting the model to the training data and ensure it can effectively learn and predict on unseen data.

K-fold Cross Validation

The validation procedure described so far assumes abundant data, allowing for a dedicated validation set separate from the training data. However, in many real-world applications, data is scarce, and using a portion of it for validation feels like a waste. K-fold cross-validation offers a solution to this by providing an accurate estimate of the model's true error rate while maximizing the usage of limited dataset [32].
In k-fold cross-validation, the original training set is partitioned into k subsets (folds) of size m=k. Each iteration, one fold is treated as a validation set, and the method is fit on the remaining k-1 folds [15]. Finally, the average of all these errors is the estimate of the true error. A special case of k-fold cross-validation occurs when k is equal to the number of data points (m). This is called leave-one-out cross-validation (LOO) and involves training on all but one data point and validating on the remaining single point. This process is repeated for all data points, providing a very thorough but computationally expensive estimate of the error rate.

3.1.2 Gaussian Regression Models

A Gaussian process is a stochastic process that is in general a collection of random variables indexed by time or space. A Gaussian Process $\mathcal{GP}$ is specified by a mean function $m(x)$ and a covariance function $k(x, x')$. Its special property is that any finite collection of these variables follows a multivariate Gaussian distribution [4]. We can think of a $\mathcal{GP}$ as a collection of infinitely many functions, all following a joint Gaussian distribution. This means that given $f(x) \sim \mathcal{GP}(m(x), k(x, x'))$, then for any set of points $x_1, x_2, \ldots, x_n$, the function values $f(x_1), f(x_2), \ldots, f(x_n)$ have a joint multivariate Gaussian distribution.

Gaussian Process Regression (GPR)

A Gaussian Process is the foundation upon which Gaussian Regression Processes (GPR) are built. This is a specific application of a Gaussian Process used for regression tasks. GPR leverages the properties of a $\mathcal{GP}$ to model the relationship between features and a target variable in your data. It assumes the data points follow a Gaussian distribution and uses the $\mathcal{GP}$ framework to make predictions and quantify uncertainty. Unlike traditional regression methods that estimate parameters for a fixed functional form (like linear or polynomial regression), GPR models the entire distribution of functions consistent with observed data. In Gaussian Process Regression (GPR), it is possible to

incorporate prior knowledge about the data into the model through the choice of the kernel function and prior distributions. The key steps in GPR are as follows:

Prior: The prior is a probability distribution that represents our initial beliefs about the functions we're trying to model. It essentially encodes our assumptions about how the features might relate to the target variable before we see any actual data. The prior fixes the properties, like mean and pointwise variance, of the functions the model considers for inference [27]. For any finite set of input points $\{x_i\}$, the corresponding function values $\{f(x_i)\}$ follow a multivariate normal distribution with mean vector $\mathbf{m}$ and covariance matrix $\mathbf{K}$.

Covariance: The covariance function $k(x, x')$, also known as the kernel function, defines how similar different data points are likely to be in terms of the target variable. It calculates the covariance between the outputs of the GPR model for two different input points. In simpler terms, it tells us how much two data points, based on their features, are expected to have similar target variable values [27]. Choosing a suitable kernel function heavily influences the types of functions the model considers. Common kernel choices include: exponential, rational quadratic and Matern kernel.

Training Data: Given training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where x_i are inputs and y_i are observed outputs, we assume $y_i = f(x_i) + \epsilon_i$, with ϵ_i being independent Gaussian noise with variance σ_n^2.

Posterior: Using Bayes' theorem, we combine the prior with the likelihood of the observed data to get the posterior distribution. The joint distribution of the training outputs $\mathbf{y}$ and the predicted values $\mathbf{f}_*$ at new inputs $\mathbf{X}_*$ is given by:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{K}_*^\top \\ \mathbf{K}_* & \mathbf{K}_{**} \end{bmatrix}\right) \tag{3.4}$$

where $\mathbf{K}$ is the covariance matrix for the training inputs, $\mathbf{K}_*$ is the covariance between training inputs and new inputs, and $\mathbf{K}_{**}$ is the covariance matrix for the new inputs.

Prediction: The conditional distribution of the predicted values given the observed data is:

$$\mathbf{f}_* | \mathbf{X}, \mathbf{y}, \mathbf{X}_* \sim \mathcal{N}(\bar{\mathbf{f}}_*, \mathbf{Cov}(\mathbf{f}_*)) \tag{3.5}$$

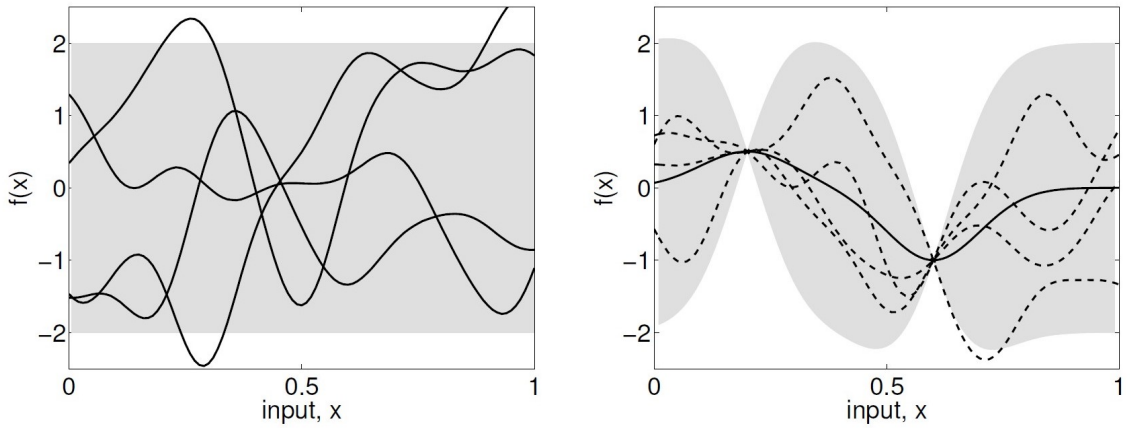


Fig. 3.1 Left figure shows four samples drawn from the prior distribution. Right figure shows the situation after two datapoints have been observed. The mean prediction is shown as the solid line and four samples from the posterior are shown as dashed lines. In both plots the shaded region denotes twice the standard deviation at each input value x. Figure from [27].

where:

$$\bar{\mathbf{f}}_* = \mathbf{K}_*(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y} \tag{3.6}$$

$$\mathbf{Cov}(\mathbf{f}_*) = \mathbf{K}_{**} - \mathbf{K}_*(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{K}_*^\top \tag{3.7}$$

Predictions are made by conditioning this posterior on new input points, yielding not only predicted values but also associated uncertainty estimates [27].

Despite its flexibility and ability to model complex relationships, GPR can become computationally expensive, especially with large datasets, due to the $\mathcal{O}(n^3)$ complexity of covariance matrix inversion [41].

3.2 Practical Work

This part of the thesis focuses on the development of a machine learning model that exploits the relationship between eye vergence and object distance. The model utilizes pupil position coordinates as input and aims to predict the corresponding gaze depth. To facilitate the development, training, and evaluation of various regression models, MATLAB's (version R2024a) Regression Learner app was employed.

Given the limited data sample, k-fold cross-validation was essential for an initial evaluation of model performance. Particular attention was given to the selection of k, as an inappropriate choice can lead to a misleading assessment of model performance. An inadequate value of k in k-fold cross-validation can lead to high variance in the evaluation scores (sensitivity to data folds) or high bias (overestimation of the model's true skill). To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using $k = 5$ or $k = 10$, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance [15].

Despite the use of k-fold cross-validation, initial tests showed overfitting in most models. To address this, 20% of the data was set aside as a holdout test set. This approach helps to ensure that the final model evaluations are unbiased and reflect the model's performance on unseen data, thus enhancing the generalizability of the regression models.

Linearized Model

In the context of the MATLAB Regression Learner, which employs Mean Squared Error (MSE) as the loss function for training models, it is important to consider the nature of the data being modeled. Given that the relationship between eye vergence and object distance is exponential, we can potentially enhance model performance by linearizing the input data. One limitation of using MSE with exponential data is that it disproportionately penalizes larger errors, making the model more sensitive to deviations at greater distances. However, our primary objective is to achieve accurate predictions for closer distances, even if this comes at the cost of less precision for farther objects. To address this, we can apply a logarithmic transformation to the distance values used in training. This transformation compresses the data range, thereby balancing the error penalties across the entire range of distances. Mathematically, if d

represents the original distance values, the transformed distances d' are obtained using:

$$d' = log(d) \tag{3.8}$$

Overall, the logarithmic transformation is a valuable preprocessing step in handling exponential relationships within regression tasks. By adopting this approach, we aim to enhance the model's predictive accuracy for near distances, aligning with our primary focus while maintaining reasonable performance for farther objects.

3.3 Results

This section presents the results obtained from training and evaluating the machine learning regression models. We analyze the performance of the various models explored, with a particular focus on the one demonstrating the best overall performance. The evaluation metrics used to assess model performance are described, followed by a detailed discussion of the results. Additionally, we discuss the influence of the choice of k in k-fold cross-validation on the final model's generalizability,.

3.3.1 Binarization Dataset

This section explores the performance of the models trained on the data obtained through the binarization method, utilizing a dataset of 590 samples. We compare the performance of a normal linear regression model with a linearized model to assess whether the linearization process, applied to address the potential issues previously discussed, can improve the model's ability to capture the underlying relationships in the data. Among the 21 models available within MATLAB's regression app, Gaussian Process Regression (GPR) models consistently demonstrated superior performance. Consequently, the following analysis will exclusively focus on the results obtained from GPR models.

The models compared in the following table are GPR models that use a rational quadratic kernel and an exponential kernel, respectively. The mathematical expressions for these kernels are as follows:

- **Rational Quadratic Kernel**:

$$k_{\mathrm{RQ}}(x, x') = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha l^2}\right)^{-\alpha}$$

where σ^2 is the signal variance, l is the length scale, and α is a positive constant that determines the relative weighting of large-scale and small-scale variations.

- **Exponential Kernel**:

$$k_{\text{Exp}}(x, x') = \sigma^2 \exp\left(-\frac{|x - x'|}{l}\right)$$

where σ^2 is the signal variance and l is the length scale.

The subsequent sections will detail the comparative performance of these GPR models, highlighting their effectiveness in capturing the complex relationships within the data.

Table 3.1 Performance of the best two GPR models with varying k-Fold values

Regression Model	RMSE (k = 5)		RMSE (k = 10)	
	Validation	Test	Validation	Test
Rational quadratic GPR	22.84	17.30	20.81	24.01
Exponential GPR	23.36	19.03	21.53	23.94
Lin. Rational quadratic GPR	27.84	25.46	25.43	28.89
Lin. Exponential GPR	27.88	28.41	26.07	29.45

The table compares the Root Mean Squared Error (RMSE) performance of the models with $k = 5$ and $k = 10$ for both validation and test sets. For $k = 5$, the MSE in validation is higher than in testing, which is unusual and can be attributed to the random splitting of the data and the presence of many duplicates in the dataset. In fact, when the dataset contains many duplicate entries, these duplicates can end up being split between the training, validation, and test sets. If the test set includes many data points that are very similar (or identical) to those in the training set, the model will perform well on the test set because it has effectively "seen" those examples before during training. This can result in a lower MSE on the test set compared to the validation set, where the duplicates might be distributed differently, leading to less overlap and thus higher MSE. Conversely, for $k = 10$, the MSE in validation is lower than for $k = 5$, but the MSE in testing is the highest, indicating overfitting. This suggests that the model performs well on the training data but fails to generalize to unseen data. Contrary to our expectations, the linearized model performs worse, as evidenced by a higher error compared to the standard model. To better appreciate the effect of linearization, it is more informative to resort to graphical representations, which can more clearly illustrate performance improvements at closer distances.

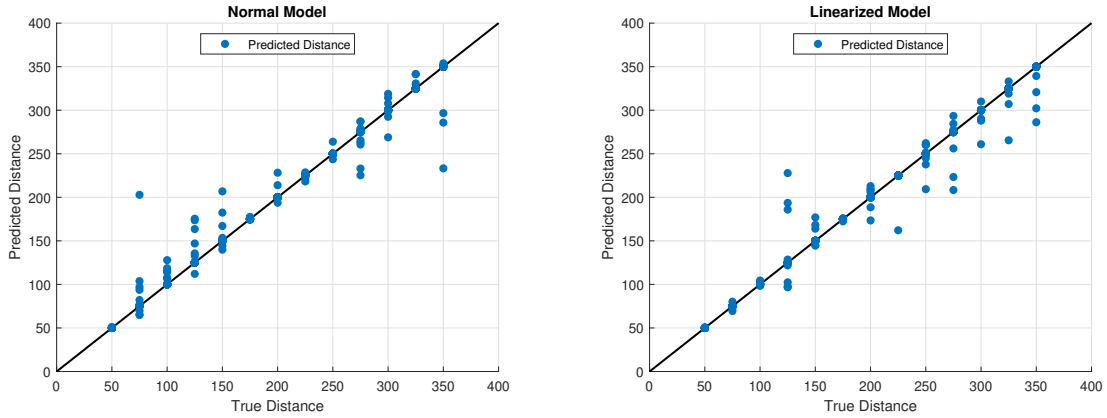


Fig. 3.2 Comparison of validation predictions between the standard model and the linearized model for $k = 10$ in k-fold cross-validation.

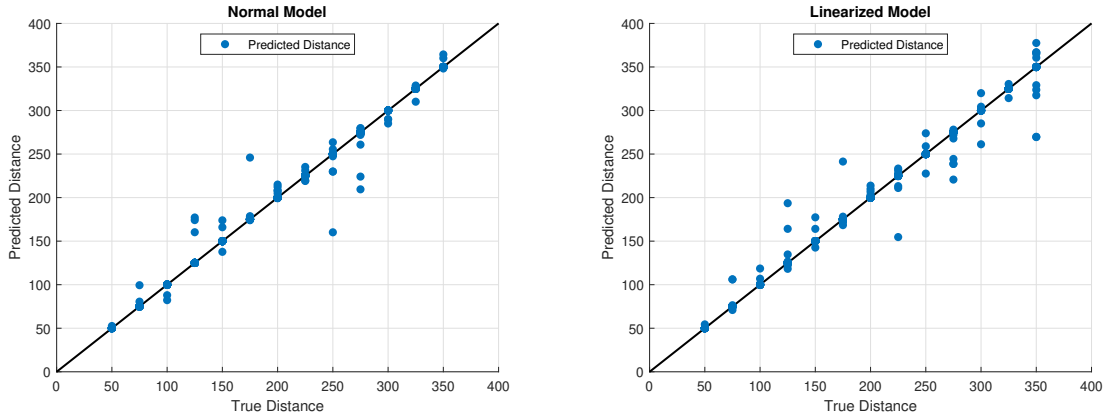


Fig. 3.3 Comparison of validation predictions between the standard model and the linearized model for $k = 5$ in k-fold cross-validation.

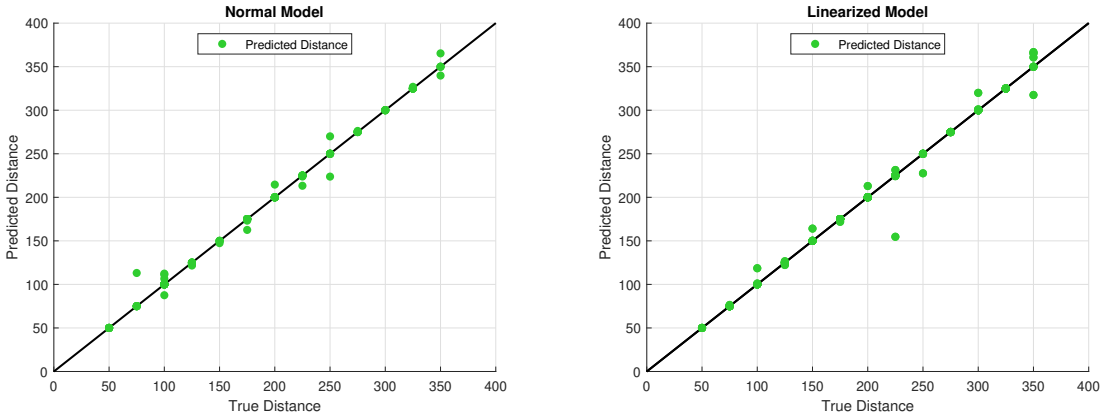


Fig. 3.4 Comparison of test predictions between the standard model and the linearized model for $k = 5$ in k-fold cross-validation.

The plots of the validation and test predictions reveal the impact of linearization, which enhances model performance at closer distances as expected. However, evaluating the models' actual performance from the plots is challenging due to the presence of many duplicate points that overlap, making it difficult to discern individual predictions. Additionally, the outliers present in the data tend to stand out more prominently, which can skew the visual interpretation of the models' effectiveness.

3.3.2 MATLAB Dataset

Here, we explore GPR models trained on the dataset processed using the MATLAB method. In contrast to the previous dataset, this one is much more compact, with just 315 entries. As in the previous section, the focus remains on investigating the impact of linearization on model performance. The following table compares the performance of GPR models utilizing two different kernels: a rational quadratic kernel and a Matern 5/2 kernel. The mathematical expressions for the latter is:

- **Matérn 5/2 Kernel**:

$$k_{\text{Matérn 5/2}}(x, x') = \sigma^2 \left(1 + \frac{\sqrt{5}|x - x'|}{l} + \frac{5(x - x')^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}|x - x'|}{l}\right)$$

  where σ^2 is the signal variance and l is the length scale.

Table 3.2 Performance of the best two GPR models with varying k-Fold values

Regression Model	RMSE ($k = 5$)		RMSE (k = 10)	
	Validation	Test	Validation	Test
Rational quadratic GPR	27.01	21.01	27.58	26.79
Matern 5/2 GPR	26.91	21.26	27.74	23.86
Lin. Rational quadratic GPR	31.74	26.26	26.53	23.48
Lin. Matern 5/2 GPR	30.8	26.83	26.04	28.77

The table obtained using the MATLAB dataset shows results similar to the previous analysis. However, for $k = 10$, the RMSE performance is worse even in the validation set. This can be attributed to the smaller sample size of this dataset, which is about half the size of the binarization one. Consequently, having more but smaller k-folds may fail to represent the entire data distribution adequately, leading to poorer model performance.

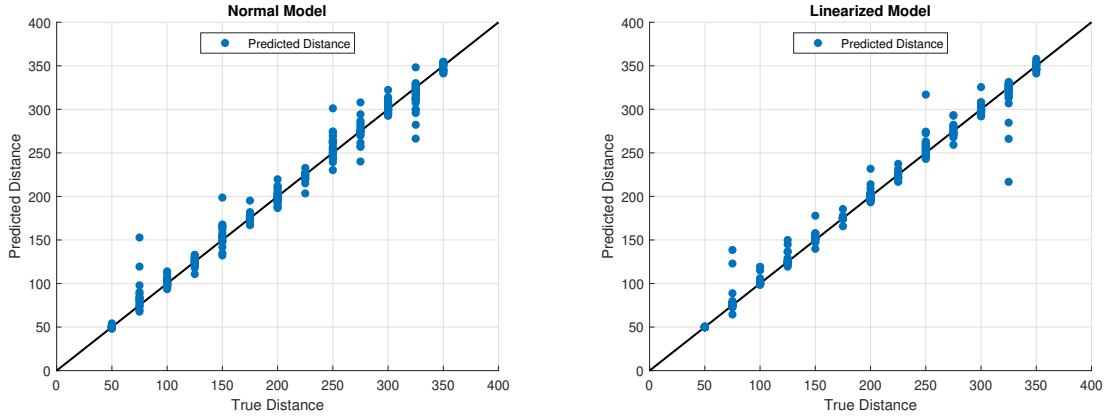


Fig. 3.5 Comparison of validation predictions between the standard model and the linearized model for $k = 5$ in k-fold cross-validation.

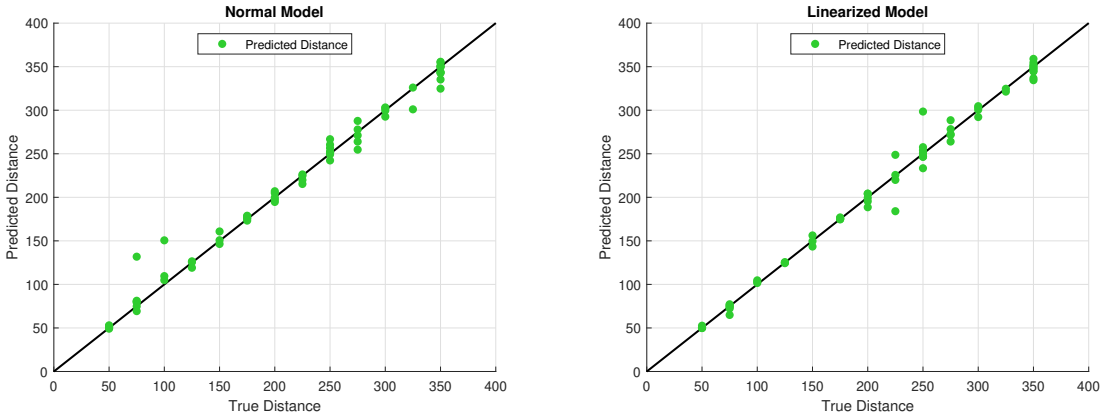


Fig. 3.6 Comparison of test predictions between the standard model and the linearized model for $k = 5$ in k-fold cross-validation.

3.4 Second Experiment and Results

A similar experiment to the one presented in the previous chapter was repeated with some modifications. This time, the distances were reduced to 1 meter, 2 meters, and 4 meters. Moreover, the videos were captured not only with the target aligned frontally but also while the observer was looking to the left and right, at an angle of approximately 25°. To achieve this, instead of moving the target, the observer rotated their body and head while maintaining their gaze fixed on the target, exploiting the vestibulo-ocular reflex. This setup aimed to examine the effect of different gaze directions on the observed pupil distances.

The modifications allowed for an expanded dataset, incorporating different gaze directions, which could provide more comprehensive insights into the relationship between eye vergence and object distance under various viewing conditions. To extract the pupil position it was used the binarization method, as it had previously proved to give the best results.

Table 3.3 Data sample distribution obtained from the experiment

Target Distance	Target Relative Position		
	Left	Frontal	Right
1 meter	33	45	37
2 meter	40	34	36
4 meter	44	50	57

Following the same methodology as before, new Gaussian Process Regression (GPR) models were trained using this updated dataset. As with the previous experiment, 20% of the data was kept as a holdout test set to evaluate model performance. A value of $k = 5$ for k-fold cross-validation was again found to be the most effective in providing a reliable estimate of model performance.
The results for the two models compared are as follows:

1. **Rational Quadratic GPR:**
    - RMSE on Validation Set: 12.62 cm
    - RMSE on Test Set: 8.43 cm
2. **Linearized Rational Quadratic GPR:**
    - RMSE on Validation Set: 14.67 cm
    - RMSE on Test Set: 9.13 cm

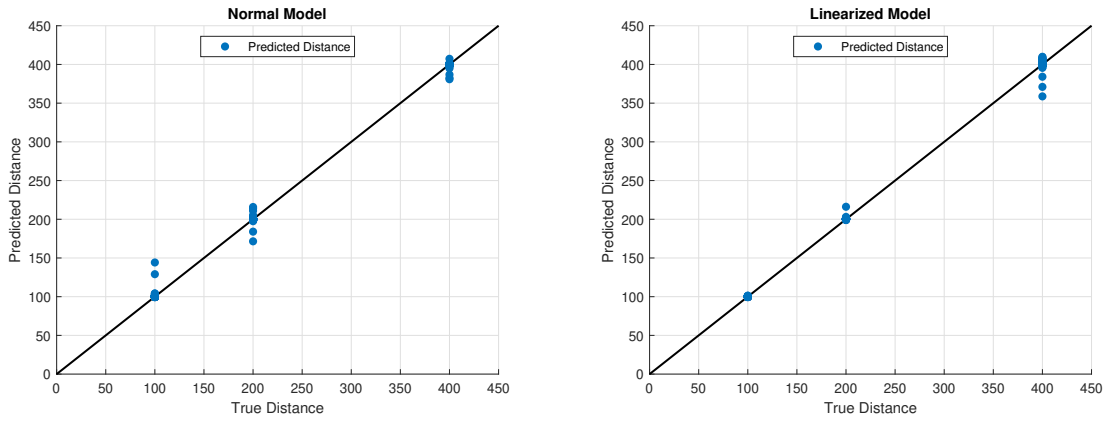


Fig. 3.7 Comparison between the normal model and the linearized version, with $k = 5$.

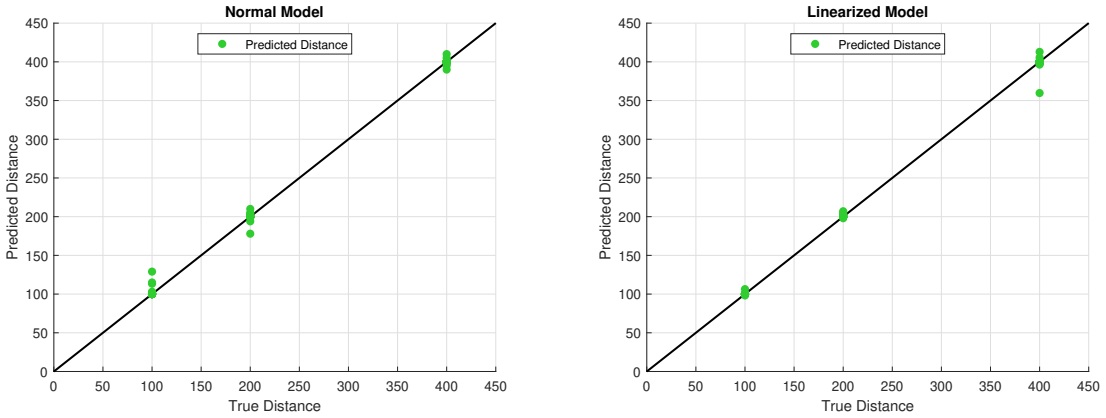


Fig. 3.8 Comparison between the normal model and the linearized version, with $k = 5$.

As can be seen from the figures, the linearized model exhibits the anticipated behavior: it is able to make more accurate predictions at closer distances, while at the expense of reduced precision for farther distances. This is a desirable outcome given the primary objective of the experiment, which prioritizes accuracy for near distances over far ones. The linearization process successfully balances the penalty for errors across the range of distances, in turn enhancing the model's performance where it is most critical.

Chapter 4

Real-Time Depth Estimation

In this chapter, we evaluate the real-time performance of the developed regression model for estimating gaze depth. To achieve this, we will decode video streams from Tobii Pro Glasses 3 in real-time, extract individual frames, detect the pupil coordinates, and use these coordinates as inputs to the regression model to predict the corresponding gaze depth. This process involves several critical steps: efficient video decoding, accurate and rapid pupil detection, and seamless integration of the model's predictions. The chapter will detail the implementation of these steps, discuss the challenges encountered, and present the real-time testing results, highlighting the model's performance in dynamic, practical scenarios.

4.1 Video Coding Concepts

Compression refers to the process of reducing the amount of data by compacting it into fewer bits. Video compression, or video coding, involves transforming digital video into a format that is more suitable for transmission or storage while typically reducing the amount of bits. This process involves two complementary systems: a compressor (encoder) and a decompressor (decoder). The encoder reduces the bit count by converting the source data into a compressed form, which is then transmitted or stored. The decoder, in turn, reconstructs this compressed form back into a representation of the original video data. Together, the encoder and decoder are referred to as CODEC [30].

Video compression is a process that reduces the size of video files by eliminating redundant information. Most video coding methods exploit both temporal and spatial redundancy to achieve efficient compression. Temporal redundancy arises because successive frames captured in close temporal proximity are often highly correlated,

especially at high frame rates. This means that frames following each other in time typically contain similar visual information, which can be efficiently encoded by referencing changes from one frame to the next, rather than storing each frame independently [33]. Spatial redundancy, on the other hand, occurs because neighboring pixels within the same frame tend to have similar values. This correlation between adjacent pixels allows for compression within a single frame by encoding the differences between neighboring pixels rather than the absolute values of each pixel. By leveraging these redundancies, video compression techniques can significantly reduce the amount of data required to represent the video.

Compression methods can be categorized as either lossy or lossless. Lossless compression retains all original data, ensuring that the decompressed video is identical to the original. This method is essential for applications requiring exact replication of the original data, such as medical imaging or archival storage. However, lossless compression typically achieves lower compression ratios compared to lossy methods. Lossy compression, in contrast, achieves higher compression ratios by discarding some data deemed less critical to the overall visual experience. This results in a reduction of file size at the expense of a slight loss in quality, which is often imperceptible to the human eye. By removing less important information, lossy compression methods like H.264 can drastically reduce file sizes. This is crucial for real-time applications, such as our gaze depth estimation system, where large, uncompressed video files would be impractical due to their substantial storage requirements and transmission bandwidth.

4.1.1 The Need for Video Encoding

The primary need for video encoding arises from the massive amount of data generated by raw video. High-resolution video streams, such as those captured by the Tobii Pro Glasses 3, consist of numerous frames per second, each containing a significant amount of pixel data. Without encoding, storing, or transmitting this raw data would require extensive disk space and bandwidth, leading to inefficiencies and potential bottlenecks in real-time processing systems.

Video encoding compresses video data using various algorithms to represent the video more compactly. The process typically involves several steps and is managed by three main functional units within a video encoder.

1. **Prediction model:** The prediction model aims to reduce redundancy by exploiting similarities between neighboring video frames (temporal redundancy) and between neighboring pixels within a frame (spatial redundancy). In H.264/AVC,

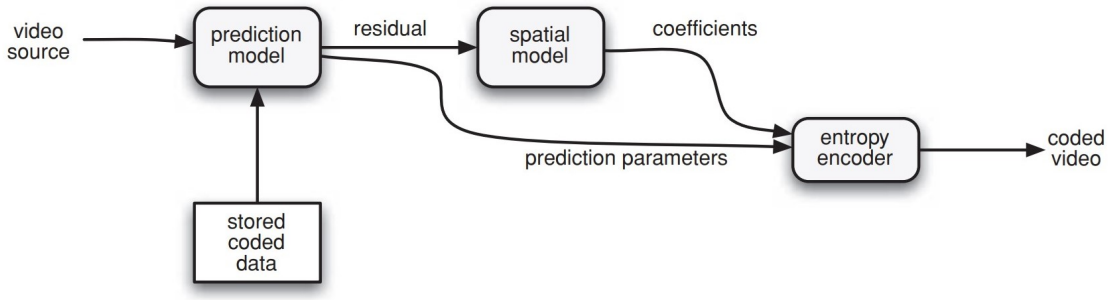


Fig. 4.1 Video encoder block diagram. Figure from [30].

predictions can be formed from data within the current frame or from one or more previous and/or future frames. The model uses techniques such as intra prediction (spatial extrapolation from neighboring samples) and inter prediction (compensating for motion differences between frames). The output of the prediction model is a residual frame, which is created by subtracting the predicted frame from the actual current frame, along with a set of model parameters.

2. **Spatial model:** The spatial model works on the residual frame produced by the prediction model to further reduce spatial redundancy. In H.264/AVC, this is achieved by applying a transform, typically the Discrete Cosine Transform (DCT), to the residual samples. This transformation converts spatial domain data into frequency domain coefficients. These coefficients are then quantized to remove insignificant values, leaving a more compact representation of the residual frame. The quantized transform coefficients are the output of the spatial model.

3. **Entropy encoder:** The entropy encoder compresses the parameters from the prediction model (such as intra prediction modes and motion vectors) and the quantized transform coefficients from the spatial model. This process removes statistical redundancy by using methods like Huffman coding or Arithmetic coding, which assign shorter binary codes to more frequent values and longer codes to less frequent ones [5]. The result is a compressed bitstream or file that can be transmitted or stored. The bitstream contains coded prediction parameters, residual coefficients, and necessary header information.

Different video codecs, such as H.264, H.265, and VP9, employ these principles to varying extents, balancing compression efficiency and computational complexity. Since Tobii Pro Glasses 3 captures video in H.264 format [38], let's take a closer look at the inner workings of this codec.

4.1.2 H.264 Video Encoding

H.264, also known as Advanced Video Coding (AVC), is one of the most widely used video compression standards. Developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC JTC1 Moving Picture Experts Group (MPEG), H.264 provides high compression efficiency and good video quality, making it ideal for a variety of applications, from streaming and broadcasting to storage and real-time video processing. Compared with previous standards, H.264 achieves up to 50% improvement in bit-rate efficiency [6].

H.264 employs a sophisticated prediction model that includes both intra-frame and inter-frame compression techniques.

- **Intra-frame (Spatial Prediction):** Each frame is divided into smaller blocks called macroblocks (16x16 or 4x4 pixels [29]). The idea is to predict the pixel values of a block based on the pixel values of neighboring, previously-coded blocks within the same frame. The prediction block is then subtracted from the actual block to form a residual, which is further processed and encoded. The use of various block sizes and prediction modes allows the encoder to adapt to different regions of the frame, balancing compression efficiency and video quality.

- **Inter-frame (Temporal Prediction):** Inter-frame compression exploits temporal redundancy by encoding differences between successive frames. Motion estimation is performed to find blocks in the current frame that match blocks in a previous frame (reference frame). Inter-frame prediction uses a range of block sizes (from 16x16 down to 4x4 [29]). The motion vectors and the difference (residual) between the blocks are encoded. This allows significant data reduction since only changes between frames need to be encoded.

Types of Frames in H.264

From this process we can distinguish three types of frames: I-frames, P-frames, and B-frames [20], each serving different roles in the compression process.

- **I-Frames (Intra-coded frames):**I-frames are encoded using only intra-frame compression. They do not reference any other frames and can be decoded independently. I-frames serve as key frames, providing reference points for decoding subsequent P-frames and B-frames. They are essential for random access and error recovery in video streams.

- **P-Frames (Predictive-coded frames):**P-frames use inter-frame compression, referencing previous I-frames or P-frames to encode the changes (motion vectors and residuals). They require less data to encode compared to I-frames, as they exploit temporal redundancy. P-frames rely on previously decoded frames, meaning they cannot be decoded independently.
- **B-Frames (Bi-predictive-coded frames):**B-frames use both previous and subsequent frames (I-frames or P-frames) as references for encoding. This allows for even higher compression efficiency by exploiting temporal redundancy more effectively. Like P-frames, B-frames cannot be decoded independently and rely on multiple reference frames.

By using a combination of I, P, and B frames, H.264 achieves high compression efficiency. I-frames provide key reference points, while P-frames and B-frames significantly reduce the amount of data needed to represent motion and changes between frames [12]. This layered approach enables H.264 to deliver high-quality video at lower bit rates, making it an ideal choice for applications requiring real-time processing and transmission, such as our gaze depth estimation system.

4.2 Video Decoding in Real-Time

In real-time systems, video encoding must be rapid to keep up with the incoming stream. The Tobii Pro Glasses 3 use H.264 encoding to efficiently compress the captured video, allowing for manageable transmission. They support live streaming of the scene camera video via the Real-Time Streaming Protocol (RTSP).

4.2.1 Real-Time Streaming Protocol

Real-Time Streaming Protocol (RTSP), is an application-level network control protocol designed for use in entertainment and communications systems to control streaming media servers. The protocol is used for establishing and controlling media sessions between endpoints. RTSP is used for various purposes including media streaming applications such as live video and audio.

RTSP offers several key features that make it suitable for streaming media. One of its primary functions is to provide control commands such as play, pause, and stop, allowing users to manage the media stream similarly to how they would use a remote control. RTSP operates within a client-server architecture, where the client sends

requests and the server responds, facilitating efficient communication and control over the media stream [31]. Additionally, RTSP is transport protocol independent, meaning it can function with various transport protocols like UDP, TCP, or others, providing flexibility in data transmission.

Socket

A socket is a software construct that enables communication between different processes on the same or different machines. It acts as an interface between the application layer and the transport layer within the network stack [17]. In the context of real-time video streaming and RTSP, a socket is an endpoint for sending and receiving data across a network. Sockets allow for communication between two devices over a network using protocols such as Transmission Control Protocol (TCP) or User Datagram Protocol(UDP).

- **TCP Sockets:** These provide reliable, connection-oriented communication. Data sent via TCP is guaranteed to arrive in the same order it was sent. This type of socket is often used for the control connection in RTSP to ensure reliable communication for setup, control messages, and metadata exchange.
- **UDP Sockets:** These provide connectionless communication, which doesn't guarantee order or even delivery of data packets. However, this trade-off allows for low latency and continuous streaming, crucial for a smooth user experience. UDP prioritizes speed over reliability, making it ideal for streaming applications like video and audio. This is because these applications can tolerate some packet loss, and uninterrupted delivery is more important than ensuring every single packet arrives.

4.2.2 Connecting to the Glasses

In this setup, RTSP is used to initiate and control the video stream from Tobii Pro Glasses 3. The client sends a series of RTSP commands to negotiate the streaming parameters and initiate the video feed. These commands are sent through a TCP socket, ensuring reliable communication and setup of the stream. During the RTSP setup phase, the client and server negotiate the use of RTP (Real-Time Protocol) for transporting the video stream. In this phase it is also specified that the RTP packets will be sent over UDP.

In H.264 video streaming over RTP, the video data is packetized into RTP packets for transmission. An RTP packet consists of two main parts: the RTP header and the RTP payload. The H.264 encoder compresses raw video frames into compressed video frames, which are then segmented into smaller pieces to fit into these packets [40]. The header, which is usually 12 bytes long, contains information like timestamps or the sequence number, which enables the receiver to detect any lost packets and restore the correct order of the packets. On the other hand, the payload of an RTP packet contains one or more Network Abstraction Layer (NAL) units.

NAL units are the fundamental building blocks of the H.264 encoded video stream, each containing a portion of the video data. These NAL units can represent various types of video information. A Sequence Parameter Set (SPS) contains important parameters needed to decode a series of video frames, such as resolution, frame rate, and other sequence-level settings. A Picture Parameter Set (PPS) includes information necessary to decode individual pictures within a sequence. It works alongside the SPS to provide the complete set of parameters required for decoding, such as entropy coding mode and slice group mapping. Instantaneous Decoding Refresh (IDR) frames are a special type of I frame. Like I frames, they are self-contained and can be decoded independently. However, IDR frames also signal the end of a GOP (Group of Pictures). When an IDR frame is received, the decoder can discard all previously decoded data and begin anew from this frame, ensuring synchronization and recovery from data loss. In other words, any frame that follows an IDR frame in decoding order cannot reference any frame that precedes the IDR frame. Non-IDR frames, on the other hand, are regular frames that follow IDR frames. They rely on previously decoded frames for reconstructing the video image. Finally, Fragmentation Units (FU-A, FU-B) for splitting large NAL units into smaller parts.

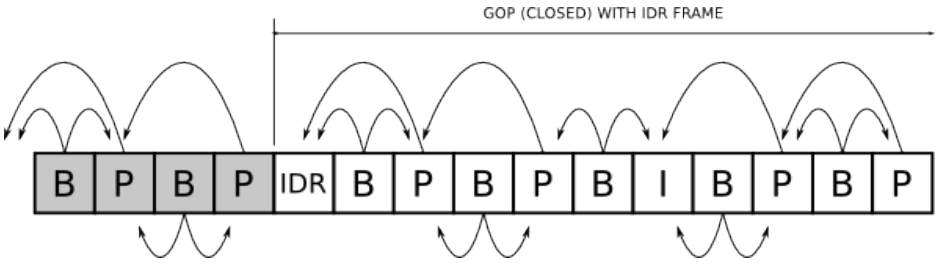


Fig. 4.2 Example of a GOP: the initial frame is always an IDR frame (image from [37]).

The digesteyepacket function processes RTP packets to extract and handle these NAL units:

1. Extracts the NAL byte from the RTP payload to determine the type of NAL unit.
2. Identifies the NAL unit type and processes it accordingly.
3. Handles different NAL unit types such as SPS, PPS, and fragmented units (FU-A, FU-B).
4. Prepares the NAL units for decoding by adding start bytes and ensuring they are in a format suitable for video decoders.

```
def digesteyepacket(st):
  nal = st[12]  # NAL byte
  typ = nal & 0b00011111
  startbytes=b"\x00\x00\x00\x01" # this is the sequence of four
      bytes that identifies a NAL packet.
  global bypass
  match typ:
  ...
  #Handling other cases
  ...
    #Reconstructing fragmented units
    case 28: # FU-A
      return rtp_h264_unpack_fu(st, 0), False
    case 29: # FU-B
      return rtp_h264_unpack_fu(st, 1), False
    #Returning True for SPS or PPS packets
    case 7: # 7 NAL unit SPS Packet
      bypass = False
      return startbytes+st[12:], True
    case 8: # 8 NAL unit PPS Packet
      bypass = False
      return startbytes+st[12:], True
    case 1: # 1 non-IDR picture
      if bypass:
        return bytes(0), False
      else:
```

```
26          return startbytes+st[12:], False
27      case _: # 1-23 other NAL units
28        return bytes(0), False
```

Listing 4.1 This function processes a UDP packet, transforming it into a format suitable for writing to disk, which can then be recognized by standard media players as an H.264 stream.

4.2.3 The Actual Decoding Process

Decoding a video stream frame by frame is not feasible due to the nature of video encoding processes. To decode the video stream in real-time, it is necessary to wait until a complete Group of Pictures (GOP) is received. The following function accomplishes this by collecting an entire GOP before decoding. After an inspection of the incoming bytes received from the Tobii Pro Glasses 3, it emerged that each GOP starts with a SPS followed by a PPS, which are identified by flags from the digesteyepacket function. Once the initial SPS and PPS are received, subsequent frames are collected until the next SPS and PPS are encountered, signaling the end of the current GOP. This complete GOP is then stored in a file, ready for decoding.

```
1  def sink_eyes():
2      header = b''
3      first_GOP = True #The first time we receive a PPS there is
           nothing to be decoded.
4      while (time.perf_counter_ns() - t_init)/1000000 < rn:
5        with open(fname_eyes,'wb') as fe:
6          if(not first_GOP):
7            fe.write(header) #writes data in the eyes file
8
9          flag = True
10         counter = 0
11         header = b''
12
13         while(flag):
14           reeyes = s4.recv(65507)  # Receive data from socket
               in chunks
15           #starting_sequence is True when receiving PPS or SPS
               units
16           data, starting_sequence = digesteyepacket(reeyes)
```

```
17            if (starting_sequence):
18              counter = counter + 1
19              header = header + data
20            if (counter == 2):
21                flag = False
22            if(flag and not starting_sequence):
23              fe.write(data) #writes data in the eyes file
24
25          first_GOP = False
26        fe.close()
27        #Here open the file and decode
28        decode_from_file("eyes.h264")
```

Listing 4.2 Sink_Eye Function: used to store a GOP in a file.

The decode_from_file function is responsible for decoding each GOP previously stored in a file. Initially, the function reads the entire video file as bytes and then starts a FFmpeg process to decode the video into raw frames. FFmpeg is a free and open-source toolset for multimedia processing, that allows to decode the H.264 video stream. A separate thread is started to handle writing the bytes from the file to the FFmpeg process's input. The main loop reads the decoded frames from the process's output, converting each frame into a NumPy array, which can then be read by OpenCV. The frames undergo pupil detection analysis, and if pupils are detected, their coordinates are forwarded to a MATLAB function for distance prediction. This involves a call to the MATLAB engine, as the regression model for distance prediction was developed using in that environment. Upon receiving the coordinates, MATLAB executes the regression model to calculate the distance. Subsequently, the calculated distance is transmitted back to the Python environment for further processing.

```
1 def decode_from_file(filename):
2
3     width, height = 1024, 256 #Frame size
4     with open(filename, 'rb') as binary_file:
5       in_bytes = binary_file.read()
6
7     #Starting the FFmpeg process
8     process = sp.Popen(shlex.split('ffmpeg -i pipe: -f rawvideo
          -pix_fmt bgr24 -an -sn pipe:'), stdin=sp.PIPE, stdout=
        sp.PIPE, bufsize=10**8)
9
```

```
10      thread = threading.Thread(target=writer, args=(process,
            in_bytes))
11      thread.start()
12
13      # Read decoded video (frame by frame)
14      while True:
15        # Read raw video frame from stdout as bytes array.
16        in_bytes = process.stdout.read(width * height * 3)
17        if not in_bytes:
18            break  # Break loop if no more bytes.
19        in_frame = (np.frombuffer(in_bytes, np.uint8).reshape([
              height, width, 3]))
20        cv2.imshow('in_frame', in_frame)
21
22        #Pupil detection
23        coordinates, detection = detect_pupils(in_frame)
24
25        #MATLAB call
26        if(detection):
27          # Convert coordinates to NumPy array
28          X = np.array(coordinates, dtype=float)
29          distance = engine.predictY2(X)
30          integer_part = int(distance // 1)
31          print(f"The distance is: {integer_part}cm")
32
33      #Thread and Process Cleanup
34      if not in_bytes:
35        thread.join()
36      try:
37        process.wait(1)
38      except (sp.TimeoutExpired):
39        process.kill()
```

Listing 4.3 Function to perform the actual real-time decoding of the video stream.

4.3 Depth Estimation Results

The gaze depth estimation system was finally ready for real-time testing. The experimental setup involved placing targets at distances of 1, 2, and 3 meters from the observer. Initial tests were conducted with the observer looking straight ahead at the targets, while subsequent tests included looking at targets positioned to the left and right to assess the system's performance in different directions. For consistency, since the binarization method was used to obtain the datasets on which the regression models were trained, it was also employed for pupil detection in these experiments. In the following table, the mean value of the gaze depth estimation obtained from these tests is presented alongside its relative error compared to the actual distances.

Table 4.1 Mean gaze depth estimation and relative error for frontal and side targets

Real Distance	Frontal		Sides	
	Mean [cm]	Error [%]	Mean [cm]	Error [%]
100 [cm]	131	31.0%	149	49.0%
200 [cm]	225	12.5%	205	2.5%
300 [cm]	271	9.6%	238	20.6%

On the left, the results of the frontal tests are displayed, while the right side shows the results of the side tests. The blue line represents the actual distance, the red line indicates the estimated gaze depth (with its mean shown as a dotted line), and the green line represents the built-in gaze depth measurement of the glasses. These visual comparisons illustrate the performance of the gaze depth estimation algorithm across the different testing scenarios.

In the frontal test, the depth estimation predictions are relatively consistent. However, when including the side view, the gaze estimation shows significant peaks, often caused by false detections during blinking. Consequently, the mean is artificially inflated. By considering the median, which is 105 cm, we obtain a value that more accurately reflects the actual distance.

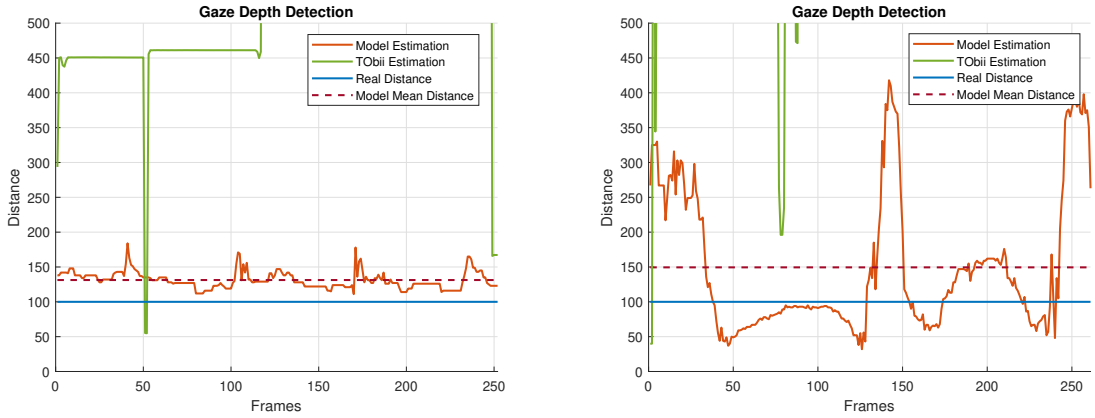


Fig. 4.3 Comparison of gaze depth estimation performance for frontal and side tests at 1 meter.

Similar to the previous results, we can observe that the estimation obtained using the regression model significantly outperforms the one provided by Tobii. The peaks in the data correspond to instances of eye blinking: during these moments, the pupil detection algorithm likely identified the pupil when the eyelid was still partially open, leading to unreliable coordinate extraction. In the case of the side view, even though the mean value closely matches the real distance, the model struggles more with estimating gaze depth. This precision overstates the actual capabilities of the model and is likely due to a particularly fortunate sample.

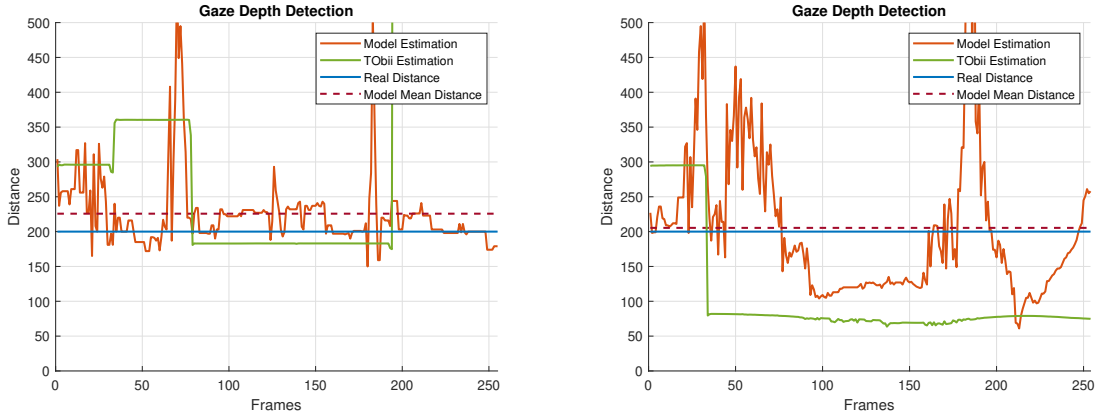


Fig. 4.4 Comparison of gaze depth estimation performance for frontal and side tests at 2 meters.

Finally, the tests done at 3 meters, show consistent results that are surprisingly precise. Gaze estimation accuracy also declines when looking at the sides compared to a frontal view. This is likely due to the pupil detection algorithm which struggles when the eyes are not looking straight forward.

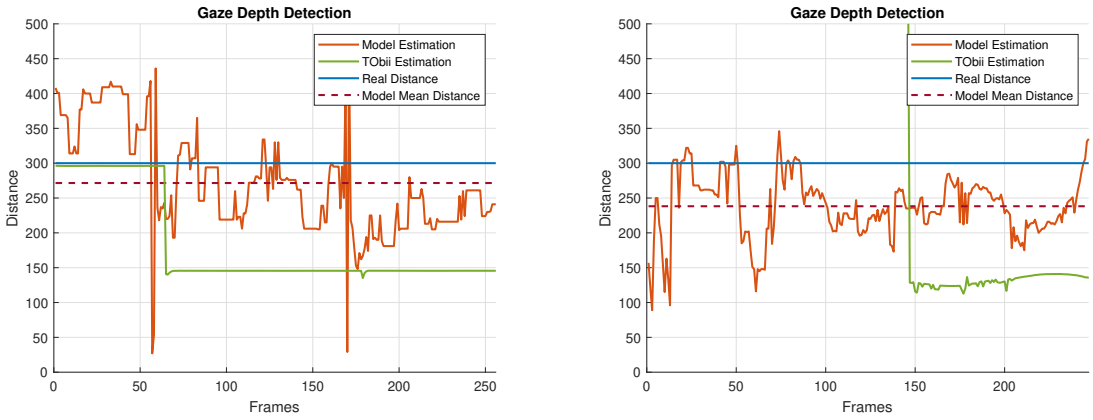


Fig. 4.5 Comparison of gaze depth estimation performance for frontal and side tests at 3 meters.

In conclusion, the developed model outperforms by far the built-in gaze depth estimation of the glasses. Contrary to our expectations, the model's performance showed a slight decline at closer distances. Imperfections in the pupil detection method itself could be a contributing factor. These limitations might be attributed to the method's sensitivity to variations in lighting conditions and other environmental factors. Alternatively, the regression model might have been overfitted due to an insufficiently large and diverse training dataset. Expanding the dataset to include a wider range of scenarios could potentially improve the model's ability to accurately handle close-range estimations. Despite the mean value providing an accurate depth value, the single estimations can oscillate very much between consecutive samples, making the model unsuitable when looking around frequently.

In addition, during the tests emerged the problem of the glasses positioning on the user. In fact when they were positioned slightly further from they eyes, the estimations obtained were completely unpredictable.

4.4 Glasses' Positioning Estimation

Accurately determining the pose of the glasses is essential, as it greatly influences the accuracy of depth estimation. By utilizing the pupil positions provided by Tobii, which includes a 3D model of the eyes detailing the pupils' positions relative to the frame, we can focus on the distance along the z-axis, as it appears to have the most significant impact. This information can then be used to train a model that incorporates this additional feature, thereby accounting for the pose of the glasses in relation to the user and improving the overall accuracy of depth estimation.

To test the feasibility of this approach, we conducted a simple experiment. With the user's gaze fixated straight ahead, the glasses were gradually moved further away from the eyes to determine if the built-in eye localization system could detect this change in position.

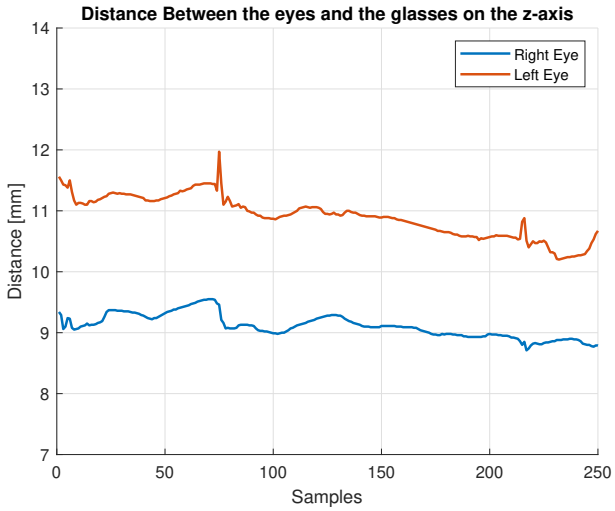


Fig. 4.6 Z-axis distance between the frame of the glasses and the eyes.

Unfortunately, as can be seen in figure 4.6, Tobii's software is unable to correctly estimate the distance. The glasses were moved by at least 1cm from their initial position, but this change is not reflected in the plot. For this reason any attempt to integrate the gaze depth estimation with a correction based on the positioning of the glasses cannot be pursued in this way.

4.4.1 Roll Angle Estimation

To estimate the roll angle of the glasses, we can apply a similar reasoning as before by utilizing Tobii's software to compute the difference between the y-coordinates of

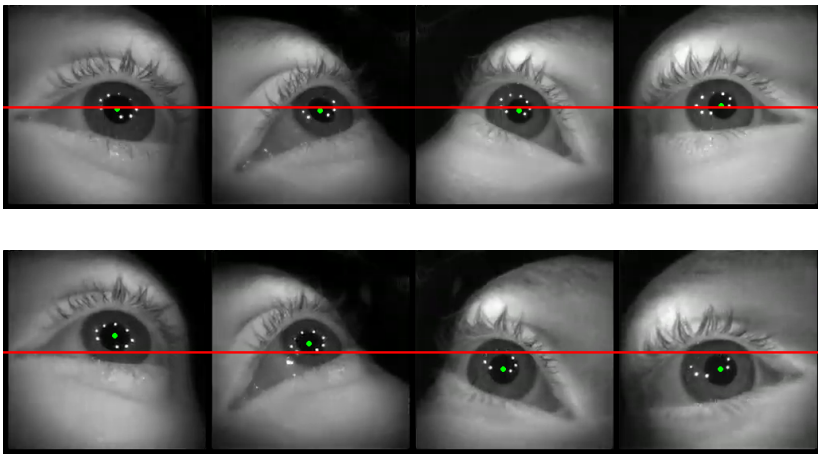


Fig. 4.7 Comparison between images of the eyes obtained with the glasses positioned correctly (top figure), and with misplaced glasses (bottom figure).

the pupils. If the glasses are positioned correctly, the eyes should be at the same height. Any discrepancy in the y-coordinates indicates a non-zero roll angle of the glasses. Specifically, since the glasses rotate precisely at the midpoint between the eyes, it means that if one eye rises by a certain amount, the other eye drops by the same amount, resulting in a total difference in their y-coordinates that corresponds to the sum of these changes.

To test the detectability of the roll angle, we conducted an experiment where the user initially positioned the glasses correctly and, after a few seconds, rotated them left and right.

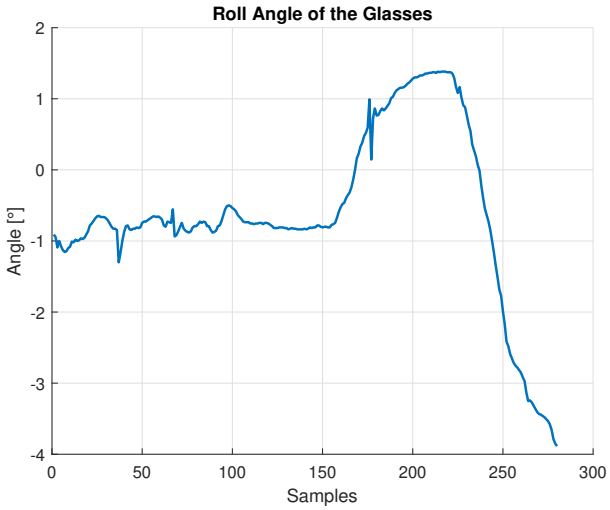


Fig. 4.8 Roll angle estimation.

As illustrated in figure 4.8, the roll angle can be successfully detected based on the relative position of the pupils. This estimated roll angle could then be used to adjust the extracted pupil coordinates, thereby enhancing the accuracy of depth estimation. It is important to consider that most individuals have natural asymmetries in their facial structure, which may result in a slight difference in the height of their eyes. Therefore, a discrepancy in the y-coordinates of the pupils might not necessarily indicate a misalignment of the glasses but rather a natural variation in facial anatomy.

Chapter 5

Conclusions

In this thesis, we explored various aspects of gaze depth estimation using Tobii Pro Glasses 3. Our key contributions include:

- **Development of a Pupil Detection Algorithm:** We developed various pupil position detection algorithms and compared their performances under varying conditions.
- **Development of a Regression Model:** We then developed a machine learning regression model to estimate gaze depth from the previously extracted pupil position coordinates.
- **Real-Time Video Decoding:** Implemented a robust pipeline for real-time decoding of H.264 video streams, enabling on-the-fly analysis and gaze depth estimation.
- **Glasses' Positioning Estimation:** Investigated the impact of the distance between the glasses and the eyes, as well as the roll angle of the glasses, on depth estimation accuracy.

Our research makes notable contributions to the field of gaze estimation. We have demonstrated that our regression model outperforms the built-in depth estimation capabilities of the glasses, offering enhanced accuracy. The methodology developed for real-time analysis and depth estimation is robust and adaptable for various applications. Moreover, our detailed analysis of glasses' positioning effects provides valuable insights that can inform future improvements in gaze tracking systems.

Despite the promising results, our research has several limitations. Natural facial asymmetries can affect the accuracy of depth estimation and glasses' positioning analysis.

Environmental factors, such as lighting conditions, also impact the performance of pupil detection algorithms. Furthermore, the generalizability of our regression model is limited by the specific dataset used for training.

Future research can build on our findings in several ways. Improving pupil detection algorithms to better handle variations in lighting and eye occlusions is a key area for development. Expanding the dataset and exploring different machine learning techniques could refine the regression model further. Integrating additional sensors, such as accelerometers or gyroscopes, may enhance the accuracy of glasses' positioning and depth estimation. Finally, developing methods for user-specific calibration can address individual facial asymmetries and improve the reliability of depth estimation.

References

[1] (2014). *The OpenCV Reference Manual*. OpenCV, 2.4.13.7 edition.

[2] Arefin, M. S., Swan II, J. E., Cohen Hoffing, R. A., and Thurman, S. M. (2022). Estimating perceptual depth changes with eye vergence and interpupillary distance using an eye tracker in virtual reality. In *2022 Symposium on Eye Tracking Research and Applications*, ETRA '22, New York, NY, USA. Association for Computing Machinery.

[3] Augusteyn, R., Nankivil, D., Mohamed, A., Maceo Heilman, B., Pierre, F., and Parel, J.-M. (2012). Human ocular biometry. *Experimental eye research*, 102:70–5.

[4] Beckers, T. (2021). An introduction to gaussian process models.

[5] Benvenuto, N. and Zorzi, M. (2011). *Principles of Communications Networks and Systems*. Wiley.

[6] Chen, J.-W., Kao, C.-Y., and Lin, Y.-L. (2006). Introduction to h.264 advanced video coding. volume 2006, pages 6 pp.–.

[7] Conti, F. (2010). *FISIOLOGIA MEDICA*. McGraw-Hill Interamericana de España S.L.

[8] Cunningham, P., Cord, M., and Delany, S. J. (2008). *Supervised Learning*, pages 21–49. Springer Berlin Heidelberg, Berlin, Heidelberg.

[9] Dodgson, N. (2004). Variation and extrema of human interpupillary distance. volume 5291, pages 36–46.

[10] Gidlöf, K., Wallin, A., Dewhurst, R., and Holmqvist, K. (2013). Using eye tracking to trace a cognitive process: Gaze behaviour during decision making in a natural environment. *Journal of Eye Movement Research*, 6.

[11] Gonzalez, R. and Woods, R. (2018). *Digital Image Processing*. Pearson Education, New York, 4th edition.

[12] Gu, X. and Zhang, H. (2003). Implementing dynamic gop in video encoding. In *2003 International Conference on Multimedia and Expo. ICME '03. Proceedings (Cat. No.03TH8698)*, volume 1, pages I–349.

[13] Iskander, J., Hossny, M., and Nahavandi, S. (2018). A review on ocular biomechanic models for assessing visual fatigue in virtual reality. *IEEE Access*, PP:1–1.

[14] Iskander, J., Hossny, M., and Nahavandi, S. (2019). Using biomechanics to investigate the effect of vr on eye vergence system. *Applied Ergonomics*, 81:102883.

[15] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

[16] Jones, J., Edewaard, D., Tyrrell, R., and Hodges, L. (2016). A schematic eye for virtual environments. pages 221–230.

[17] Kalita, L. (2014). Socket programming. *International Journal of Computer Science and Information Technologies*, 5(3):4802–4807. Assam Down Town University, Guwahati, India.

[18] Kandel, E. R., Schwartz, J. H., and Jessell, T. M., editors (1991). *Principles of Neural Science*. Elsevier, New York, third edition.

[19] Kim, J.-S., An, B. H., Jeong, W.-B., and Lee, S.-W. (2021). Estimation of interpupillary distance based on eye movements in virtual reality devices. *IEEE Access*, 9:155576–155583.

[20] Koumaras, H., Skianis, C., Gardikis, G., and Kourtis, A. (2005). Analysis of h.264 video encoded traffic.

[21] Mackey, D. A. (2022). What colour are your eyes? teaching the genetics of eye colour & colour vision. edridge green lecture rcophth annual congress glasgow may 2019. *Eye*, 36(4):704–715.

[22] Malmivuo, J. and Plonsey, R. (1995). *Bioelectromagnetism - Principles and Applications of Bioelectric and Biomagnetic Fields*.

[23] Miller, B., Keane, C., and O'Toole, M. (1997). *Miller-Keane Encyclopedia & Dictionary of Medicine, Nursing & Allied Health*. ENCYCLOPEDIA AND DICTIONARY OF MEDICINE, NURSING, AND ALLIED HEALTH. Saunders.

[24] Mlot, E. G., Bahmani, H., Wahl, S., and Kasneci, E. (2016). 3d gaze estimation using eye vergence. In *International Conference on Health Informatics*.

[25] Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press.

[26] O'Rahilly, R. and Müller, F. (1983). *Basic Human Anatomy: A Regional Study of Human Structure*. Saunders.

[27] Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning*. The MIT Press.

[28] Reichelt, S., Haeussler, R., Fütterer, G., and Leister, N. (2010). Depth cues in human visual perception and their realization in 3d displays. volume 7690.

[29] Richardson, I. (2007). An overview of h. 264 advanced video coding. *Vcodex white paper*.

[30] Richardson, I. (2010). The h.264 advanced video compression standard: Second edition.

[31] S, J. (2006). Secure real-time streaming protocol (rtsp) for hierarchical proxy caching.

[32] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press.

[33] Shi, Y. and Sun, H. (2019). *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards.* Image processing series. CRC Press, Taylor & Francis Group.

[34] "The MathWorks Inc." (2024). imfindcircles. https://it.mathworks.com/help/images/ref/imfindcircles.html.

[35] The MathWorks, Inc. (2024). Machine learning in matlab. https://it.mathworks.com/help/stats/machine-learning-in-matlab.html.

[36] "The MathWorks Inc." (2024). Types of morphological operations. https://it.mathworks.com/help/images/morphological-dilation-and-erosion.html.

[37] Tiliam, Inc. (2015). Effective use of long gop video codecs. http://tiliam.com/Blog/2015/07/06/effective-use-long-gop-video-codecs/. Accessed: 2024-06-04.

[38] Tobii Technology AB (2024). *Tobii Pro Glasses 3 Developer Guide.* Accessed: 2024-06-01.

[39] Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA.

[40] Wenger, S., Hannuksela, M., Stockhammer, T., Westerlund, M., and Singer, D. (2005). Rtp payload format for h.264 video.

[41] Zhu, H., Williams, C. K. I., Rohwer, R., and Morciniec, M. (1997). Gaussian regression and optimal finite dimensional linear models. Technical report, Birmingham.