Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Robot reading human head pose and gaze direction

Bachelor's Thesis

2023
Dmytro Herashchenko

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Robot reading human head pose and gaze direction

Bachelor's Thesis

Study Programme:	Applied Informatics
Field of Study:	Informatics
Department:	Department of Computer Science
Supervisor:	prof. Ing. Igor Farkaš, Dr.

Bratislava, 2023
Dmytro Herashchenko

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Dmytro Herashchenko
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Robot reading human head pose and gaze direction
Robotická predikcia pozície hlavy a zamerania pohľadu človeka

Anotácia: Interakcia medzi človekom a robotom (HRI) výrazne závisí od schopnosti robota sledovať ľudské správanie, aby ho robot mohol predpovedať a podľa toho konať. Túto schopnosť možno uľahčiť použitím dodatočného hardvéru, ako je napríklad sledovač očí namontovaný na hlave.

Cieľ: 1. Preštudujte si literatúru o existujúcich modeloch odhadu polohy hlavy a extrakcie oka a trénujte hlbokú neurónovú sieť na predpovedanie smeru pohľadu.
2. Na trénovanie modelu použite reálne aj syntetické dáta a porovnajte výsledky presnosti.

Literatúra: Kerzel M. et al. (2017). NICO — Neuro-inspired companion: A developmental humanoid robot platform for multimodal interaction. In IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), doi:10.1109/ROMAN.2017.8172289
Palinko O. et al. (2016) A Robot Reading Human Gaze: Why Eye Tracking Is Better Than Head Tracking for Human-Robot Collaboration. IROS, pp. 5048-5054.

Vedúci: prof. Ing. Igor Farkaš, Dr.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 05.09.2022

Dátum schválenia: 05.09.2022 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Dmytro Herashchenko
Study programme: Applied Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study: Computer Science
Type of Thesis: Bachelor´s thesis
Language of Thesis: English
Secondary language: Slovak

Title: Robot reading human head pose and gaze direction

Annotation: Human-robot interaction (HRI) depends significantly on robot's ability to track human behavior in order to allow the robot to make predictions and act accordingly. This ability can be facilitated by using additional hardware, such as head-mounted eye tracker.

Aim: 1. Study the literature on existing models of head pose estimation and eye extraction, and train a deep neural network for predicting human eye gaze.
2. For training the model use real as well as synthetic data and compare the accuracy results.

Literature: Kerzel M. et al. (2017). NICO — Neuro-inspired companion: A developmental humanoid robot platform for multimodal interaction. In IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), doi:10.1109/ROMAN.2017.8172289
Palinko O. et al. (2016) A Robot Reading Human Gaze: Why Eye Tracking Is Better Than Head Tracking for Human-Robot Collaboration. IROS, pp. 5048-5054.

Supervisor: prof. Ing. Igor Farkaš, Dr.
Department: FMFI.KAI - Department of Applied Informatics
Head of department: doc. RNDr. Tatiana Jajcayová, PhD.

Assigned: 05.09.2022

Approved: 05.09.2022 doc. RNDr. Damas Gruska, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: I would like to thank my supervisor prof. Ing. Igor Farkaš, Dr., for all the invaluable help and guidance, that paved the way to successful completion of the thesis.

Abstrakt

Táto práca sa zaoberá využitím neurónových sietí na odhadovanie pohľadu očí s cieľom vytvoriť spoľahlivý systém, ktorý možno použiť na interakciu človeka a robota. Ukázalo sa, že pohľad očí je kľúčovým podnetom pre akúkoľvek sociálnu interakciu, takže poskytnutie schopnosti odhadovať ho robotom by im umožnilo viac sa podobať ľuďom. Navrhovaná metóda si nevyžaduje žiadny špeciálny hardvér ani infračervené filtre; namiesto toho využíva štandardnú RGB kameru. Táto prispôsobivosť umožňuje širšiu škálu aplikácií, a to aj pri rôznej orientácii hlavy, očí a kamery. To tiež umožňuje komukoľvek integrovať systém do svojho robota alebo nejakého iného softvéru a nebude musieť vynaložiť žiadne ďalšie peniaze na to, aby využil výhody odhadu pohľadu očí. V tomto výskume sme tiež vytvorili rôznorodý súbor údajov na odhadovanie pohľadu očí pomocou technológií Unreal Engine a Metahuman. Vďaka výkonu týchto technológií sme mohli získať prístup k modelom ľudí s vysokým rozlíšením a dokonca vytvoriť vlastné. To umožňuje meniť aj parametre, ako je vek, rasa, farba očí, tvar tváre, a vytvoriť tak rôznorodý súbor údajov na lepšiu generalizáciu modelu. Aj systém dynamického osvetlenia nám umožnil meniť svetelné podmienky, aby bol model odolnejší voči rôznym situáciám. Celkovo tieto technológie otvárajú dvere na vytváranie súborov údajov pre rôzne ďalšie problémy v oblasti strojového učenia a počítačového videnia. Použili sme aj ďalšie pretrénovane modely na úlohy, ako je segmentácia tváre a očí a odhad polohy hlavy. To nám umožnilo sústrediť sa na úlohu odhadu pohľadu očí a získať ďalšie vstupné údaje na zlepšenie výkonnosti modelov. Model však možno vylepšiť, aby sa lepšie vyrovnal s náročnými okolnosťami, ako je napríklad zlé osvetlenie alebo ľudia, ktorí nosia okuliare. Odhad pohľadu očí len pomocou kamery sa stáva náročným aj vtedy, keď sa osoba pozerá dole a kamera je nad úrovňou jej očí. Výkonnosť modelov sa hodnotila aj na rôznych datasetoch a pri testovaní v reálnom svete.

Kľúčové slová: odhad pohľadu očí, počítačové videnie, syntetický dataset, konvolučná neurónová sieť

Abstract

This thesis explores the usage of artificial neural networks for eye-gaze estimation to build a reliable eye-gaze estimation system that can be used for human-robot interaction. It has been shown that eye gaze is a crucial cue for any social interactions, so giving the ability to estimate it to the robots would allow them to be more human-like. The proposed method does not require any special hardware or infrared filters; instead, it uses a standard RGB camera. This adaptability enables a wider range of applications, even with various head, eye, and camera orientations. This, also, allows anybody to integrate the system into their robot, or some other software, and they won't have to spend any additional money to take advantage of eye gaze estimation. In this research, we have also generated a diverse dataset for eye gaze estimation using Unreal Engine and Metahuman technologies. With the power of these technologies we were able to access high resolution human models and even create our own. This also allows to change parameters like age, race, eye color, face shape to create a diverse set of data for better generalization of the model. Also the dynamic lighting system allowed us to change the lighting conditions to make the model more resistant different situations. Overall these technologies open a door to generate datasets for a variety of other problems in machine learning and computer vision. We also used other pre-trained models for tasks like face and eye segmentation, and head pose estimation. This allowed us to focus on the task of eye gaze estimation and get some additional input data to improve models performance. However, the model can be improved to better deal with challenging circumstances such as poor lighting or people wearing glasses. Estimating eye gaze just with a camera also becomes challenging when a person is looking down and the camera is above their eye level. Models performance was also evaluated on different datasets and in the real-world testing.

Keywords: eye gaze estimation, computer vision, synthetic dataset, convolutional neural network

Contents

Introduction **1**

1 Theoretical background **3**
1.1 Eye gaze estimation methods . . . 3
1.1.1 Artificial neural networks for eye gaze estimation . . . 3
1.1.2 Cross-validation . . . 4
1.2 Review of similar systems . . . 5
1.2.1 Feature-based estimation system . . . 5
1.2.2 Model-based estimation system . . . 5
1.3 Existing datasets for eye gaze estimation . . . 6
1.3.1 Columbia Gaze Data Set . . . 6
1.4 The technologies used . . . 8
1.4.1 Pytorch . . . 8
1.4.2 OpenCV . . . 8
1.4.3 RetinaFace . . . 9
1.4.4 6DRepNet . . . 9
1.4.5 Unreal Engine and Metahumans . . . 10
1.4.6 Blueprints . . . 11

2 Methodology **13**
2.1 Architecture . . . 13
2.1.1 General architecture . . . 13
2.1.2 CNN architecture . . . 14
2.2 Metahuman dataset . . . 15
2.2.1 Characters . . . 15
2.2.2 Setting up the scene . . . 16
2.2.3 Generation blueprint . . . 16

3 Implementation and testing **19**
3.1 Model iterations . . . 19
3.1.1 Iterative improvements . . . 19

3.1.2 Final model 21
3.2 Dataset testing and comparison 22
3.2.1 Fighting overfitting 22
3.2.2 Combined dataset 23
3.2.3 Dataset comparison 23
3.3 Real world testing 24

Conclusion **27**

List of Figures

1.1 Basic convolutional neural network architecture [1]. 4
1.2 Sample of Columbia gaze dataset [2]. 7
1.3 Sample of MPIIGaze dataset [3]. 7
1.4 Sample of Unity Eyes dataset [4]. 8
1.5 Example of RetinaFace library used on world's largest selfie [5]. 9
1.6 Example of 6DRepNet library used on a real image [6]. 10
1.7 Example of characters generated with MetaHuman tool [7]. 11
1.8 Blueprint editor screenshot [8]. 12

2.1 General architecture of eye gaze estimation system. 13
2.2 Architecture of the eye gaze estimation CNN. 15
2.3 The scene from Unreal Engine where the dataset was generated. 16
2.4 Sample of pictures from the generated dataset. 17

3.1 Training and validation loss comparison graph for the Metahuman Dataset 22
3.2 Training and validation loss comparison graph for the Columbia Gaze Dataset 23
3.3 Comparison matrix of cross-validated results between the three datasets 24
3.4 Example images from real-world testing with a webcam. 25
3.5 Example images from real-world testing with the NICO robot. 26

Introduction

Human-Robot Interaction (HRI) is a rapidly growing field with numerous applications in various fields such as manufacturing, healthcare, education, and entertainment. One of the most important aspects of HRI is the ability of robots to understand and respond to human behavior and intentions. Eye gaze is a key signal that people use in social interactions to convey their attention, interest, and emotions. Thus, accurate gaze estimation by robots is essential for successful HRI. Several approaches have been proposed for gaze estimation, including model-based, appearance-based, and hybrid methods. However, due to the high variability of gaze directions and head position, gaze estimation remains a challenging task.

Today, head pose estimation is often used as an approximation for eye gaze, but according to "We find that the possibility to exploit the richer information carried by eye gaze has a significant impact on the interaction. As a result, our eye tracking system allows for a more efficient human-robot collaboration than a comparable head tracking approach, according to both quantitative measures and subjective evaluation by the human participants." [9] eye gaze estimation is a better communication cue and can not be completely substituted by the head pose estimation approach.

In this paper, we are proposing the use of ANNs (Artificial Neural Networks) for predicting human eye gaze, since they have been showing spectacular results in other computer vision scenarios, outperforming all the different techniques by a mile. The proposed model is relying on other pre-trained models for finding the faces on the image and predicting head pose and then uses compiled data to make its own estimation of the eye gaze.

The aim of this thesis is to research the current eye and head pose estimation models, and implement a robust system for eye gaze estimation that can be used in robots and other applications. One of the main differences of this system from others should be the ability to work only using a normal RGB camera without any special IR filters and any additional hardware. It should also work from different positions relative to the head of the person whose eye gaze it is predicting and should work reliably despite different head positions.

Chapter 1

Theoretical background

1.1 Eye gaze estimation methods

There are two main methods of eye gaze estimation: feature-based estimation and model-based estimation. The model-based methodology relies on machine learning algorithms, including artificial neural networks, to study input features such as eye images and provide an accurate estimate of gaze direction. Effective use of model-based machine learning methods can provide significant benefits. However, this approach requires a significant amount of labeled data to train effectively and can be computationally demanding to implement, especially at the training stage. In contrast, feature-oriented methods rely on intuitive, hand-crafted features such as pupil position, and generally have lower requirements for structured data and computational power. Nevertheless, they often work best with IR cameras and lighting where the pupil becomes very bright and easy to distinguish, and become a lot less accurate when using normal RGB cameras, especially in different lighting conditions.

1.1.1 Artificial neural networks for eye gaze estimation

Artificial neural networks (ANNs) are a widely employed technique in the realm of computer vision, because of their ability to find relationships between the input and output features. They were first inspired by the human brain and the way it processes information. They are comprised of a series of neurons, that are organized into layers. The input layer of the neural network receives data that can be anything including text, images, numbers, or even binary data. The output layer of the same neural network can output completely different types of data, for example in my case the network is getting images of the eyes as inputs and then outputs numbers that correspond to angles of eyes in 3D space. The neural network architecture may consist of one or more hidden layers positioned between the input and output layers.

Various types of neural network layers exist fully connected, convolutional, recurrent,

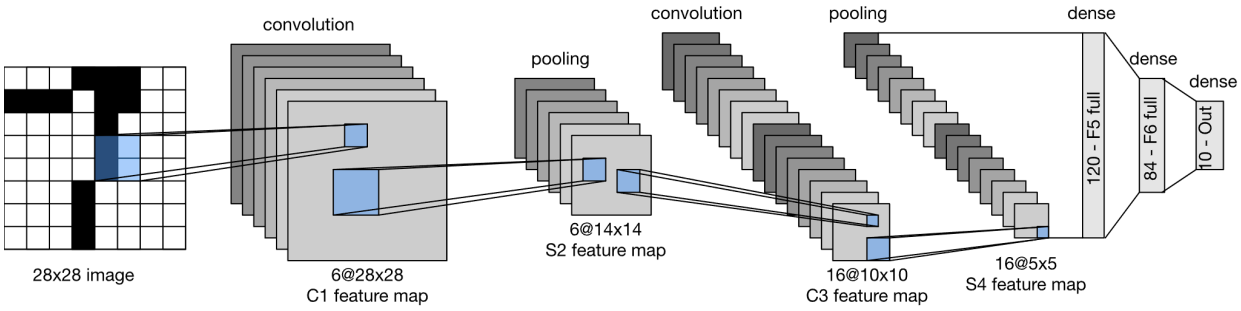

Figure 1.1: Basic convolutional neural network architecture [1].

LSTM, etc. Also, these layers can be connected in different architectures, for example, feed-forward neural networks, encoder/decoder networks, and transformers. But for computer vision-related tasks, one of the most successful architectures is a convolutional neural network. This type of network uses convolutional layers, which use utilize filters like smaller networks that go around the image and look for some specific features. Then it aggregates these features to make some more general assumptions about the image, for example, classify what object is on that image.

The most common architecture for a convolutional neural network is shown on the fig. 1.1. It is constructed from multiple convolutional layers, each of which is often followed by a pooling layer. After that, the output of the last convolutional layer is flattened out and connected to multiple fully connected layers. They are the ones responsible for using features, found by convolutional layers, to calculate the final output.

1.1.2 Cross-validation

Cross-validation is used to evaluate model performance and generalization in machine learning. Cross-validation involves partitioning the dataset into multiple subsets, or "folds", and iteratively training and assessing the model on different combinations of said folds. Cross-validation becomes more complicated when working with multiple datasets.

K-fold cross-validation randomly divides the dataset into k equal-sized folds. To validate each fold once, the model is trained on k-1 folds and tested on the remaining folds k times. Averaged values of losses are used to evaluate performance.

When using multiple datasets, the goal is to evaluate the performance of the model across these different datasets. In this case, the datasets are once again split up into k folds and then the model is trained on the same subset of folds for each dataset and each model is then tested on each training data (leftover fold from each dataset). This

allows us to construct a matrix of model evaluation results, with the model trained on each of the datasets and then tested on all of them. We can assess the generality and robustness of the model by testing it on multiple datasets this way. Also, this allows us to compare the datasets with one another and evaluate how performing each of them is when testing on others.

We will use different types of k-fold cross-validation throughout the paper to both validate different models and compare different datasets against one another.

1.2 Review of similar systems

The study of such systems is an important part of any work, as it provides an overview of the existing literature and technologies related to the topic. In this section, we will discuss previous work done in the area of robotic gaze assessment, including methods, techniques, and models that have been developed.

1.2.1 Feature-based estimation system

An example of using a feature-based estimation method for eye gaze estimation, a study [10] can be used. It was proposed as an alternative to appearance-based methods. The authors emphasized the drawbacks of appearance-based techniques, especially on low-resolution and noisy photographs taken in difficult real-world situations, such as rapid lighting changes. The proposed strategy, which competes with recent appearance-based approaches, concentrates on recognizing eye area landmarks through a single eye image to overcome these problems.

Similar to previous feature-based approaches, the strategy proposed by the authors is intended to extract rich information by including more landmarks and considering features such as iris and eye borders. The HRNet backbone network is used to learn image representations at low resolution to improve robustness to low-resolution inputs.

Our model-based approach is trying to fight the aforementioned issues while it still is an appearance-based technique

1.2.2 Model-based estimation system

As for a model-based system, one of the examples would be the network proposed in the [11] paper. Authors addressed the difficult problem of assessing human eye gaze from real-world eye images in their work. They pointed out that the unobservable nature of the center of the eyeball in 2D images makes it difficult to accurately determine the gaze direction from only eye images. As a result, it is hard to obtain a highly accurate estimation of eye gaze. The authors suggest a novel artificial neural network

architecture specifically created for the task of eye gaze estimation from a single eye input to overcome this difficulty.

In their network architecture, they incorporate an intermediate graphical representation rather than directly regressing the pitch and yaw angles of the eyeball. The 3D gaze direction estimation process becomes easier with this intermediate representation. By utilizing this strategy, the authors demonstrate higher accuracy in quantitative and qualitative evaluation when compared to existing approaches. In addition, their approach shows robustness to changes in eye gaze, head posture, and image quality.

So this paper is trying to fight the exact same problem of real-world eye gaze estimation with changing lighting conditions, head position, and noisy images that make it more difficult and unreliable. While all of these papers are trying to make the model architecture more adapted to these problems, in this paper we will try out a different approach to overcome these challenges.

1.3 Existing datasets for eye gaze estimation

There is a variety of different eye gaze datasets which are both real-life and synthesized. In this section, we will explore different datasets, their differences, and the advantages/disadvantages of each.

1.3.1 Columbia Gaze Data Set

One of the most diverse datasets that were taken in a controlled environment and are using degrees to represent pitch and yaw is the Columbia gaze dataset [2]. The 5,880 photos of 56 people that make up the dataset provide a comprehensive set of eye gaze data with different gaze directions and very importantly different head poses. In terms of the number of people, this dataset is superior to other eye gaze datasets that were publicly accessible at the time of its release. The individuals in the sample came from various ethnic backgrounds, and it is important to note that almost half of them wore glasses.

Each high-quality image in this dataset has a resolution of 5,184 $\times$ 3,456 pixels, which is great for predicting the dataset itself, but is not representative of the average camera used for these purposes, so it doesn't really help with the training model to better predict data in varying conditions. Another problem with it is that the lighting conditions are pretty much the same every time, which doesn't add any diversity as well. On the fig. 1.2 we can see a sample from the dataset with different eye and head positions.

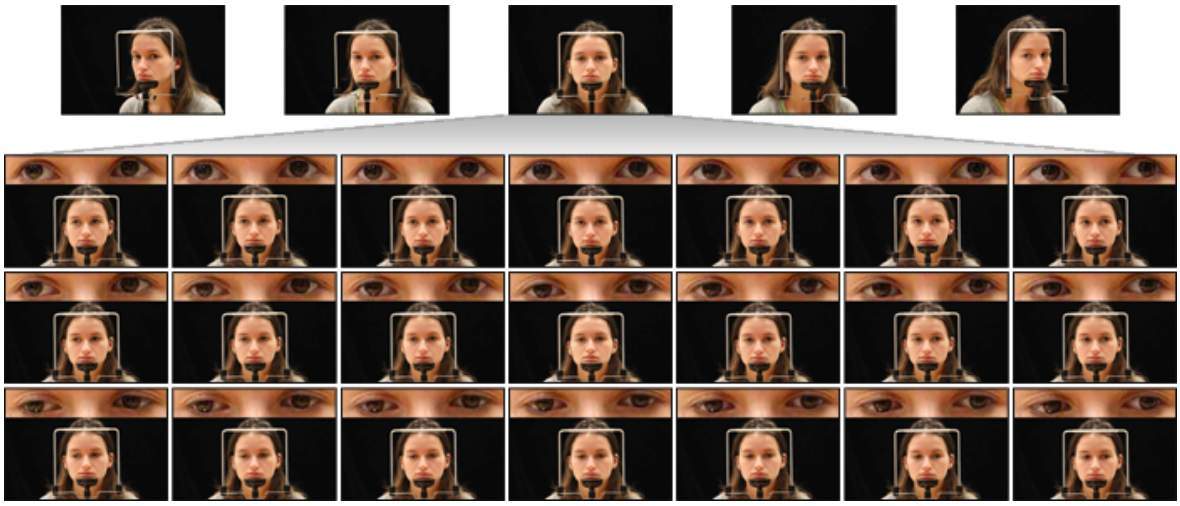

Figure 1.2: Sample of Columbia gaze dataset [2].

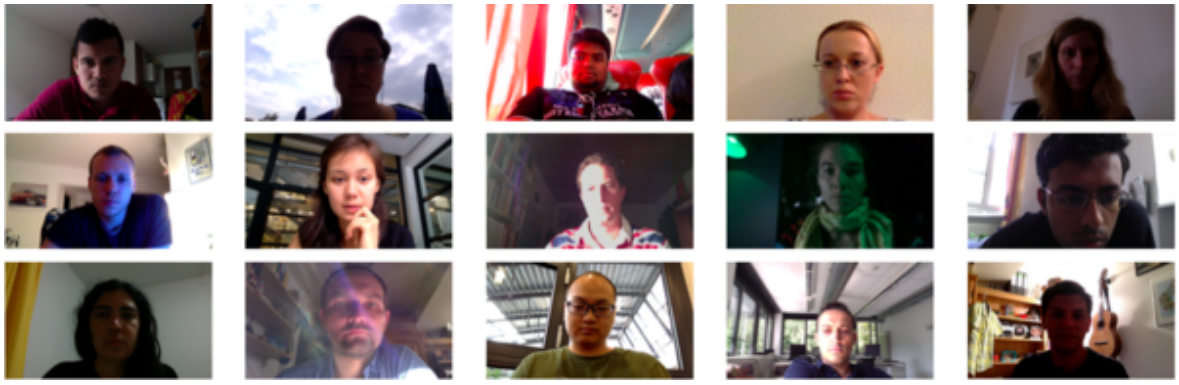

Figure 1.3: Sample of MPIIGaze dataset [3].

This is the dataset that has been chosen as the main one at first because it has a more extensive variety of people and what is critical head positions. For example, there is an MPIIGaze dataset [3] which contains nearly, 214,000 images it is only of 15 people, and they are a lot less varied in terms of head and eye positions. You can see an example of images from it in fig. 1.3. There is also a Unity eye dataset [4] which also provides the Unity eyes tool to generate different images of eyes from a high-resolution 3d model and one million images generated with the tool as a dataset. It lacks the possibility of adding accessories such as glasses, does not let us control the lighting conditions of the scene, and does not provide images of the full face since it does not exist. You can see an example of images from the dataset in fig. 1.4.

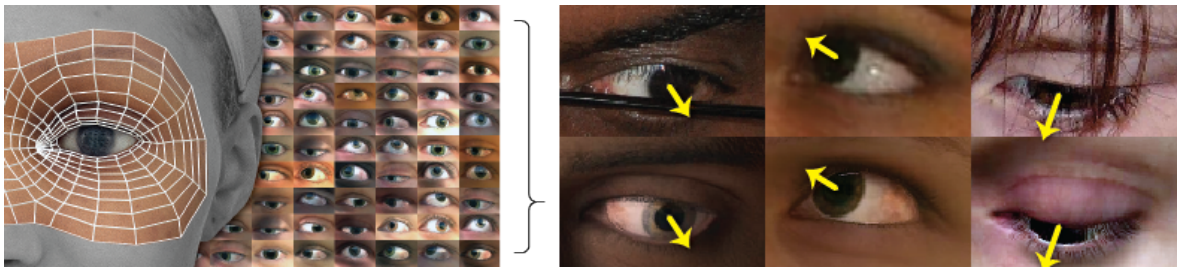

Figure 1.4: Sample of Unity Eyes dataset [4].

We will also generate our own dataset with modern game engine technology later in the paper.

1.4 The technologies used

1.4.1 Pytorch

The open-source deep learning framework, PyTorch, has become really popular in the research and business world. PyTorch, created by Facebook's AI Research team, offers a customizable and simple interface for creating and training neural networks. The framework combines automatic differentiation with dynamic computational graphics to enable fast and versatile model building. One of the key advantages of PyTorch is its ecosystem of libraries, tools, and pre-trained models that can customize and enhance its functionality. For example, Torchaudio offers audio processing capabilities, while Torchvision offers pre-trained models and tools for computer vision work. In addition, NumPy and other Python libraries for scientific computing, are easily integrated with PyTorch, further enhancing its ease of use and versatility.

PyTorch's appeal among researchers could be credited to its flexibility, which makes it easy for them to experiment with cutting-edge concepts and designs and change things on the fly. It is easier for developers to get started and get support when needed thanks to comprehensive documentation and a strong community.

1.4.2 OpenCV

OpenCV (Open Source Computer Vision software) is one of the most popular libraries for computer vision that offers a comprehensive collection of tools and functions for image processing. It has a wide range of capabilities, including the ability to edit images and videos, identify objects, and recognize features. The wide range of algorithms and methods in OpenCV makes it a great tool for computer vision tasks.

OpenCV is an essential tool for using PyTorch with images and video. Artificial neural networks can be trained and deployed using PyTorch's powerful ecosystem, and

OpenCV's effective image preprocessing and data augmentation methods are useful additions. With the help of OpenCV, you can load and prepare photos, apply filters, apply geometric transformations, and extract image features. In the deep learning process, these activities are essential.

In addition, it allows for easy real-time video processing right from the webcam and can be used to run PyTorch applications on that real-time feed. This is made possible by OpenCV's extensive support for camera interfaces, and this is what makes it perfect for our use case of real-time eye gaze estimation.

1.4.3 RetinaFace

Retina face [5] is a well-known pre-trained PyTorch model for face detection and localization in the real world with different that has shown robust performance on a variety of datasets. It provides accurate position tracking of multiple faces on the image and also finds some important locations on the faces, such as the positions of the eyes and nose. These can then be used to extract images of the eyes for eye tracking.

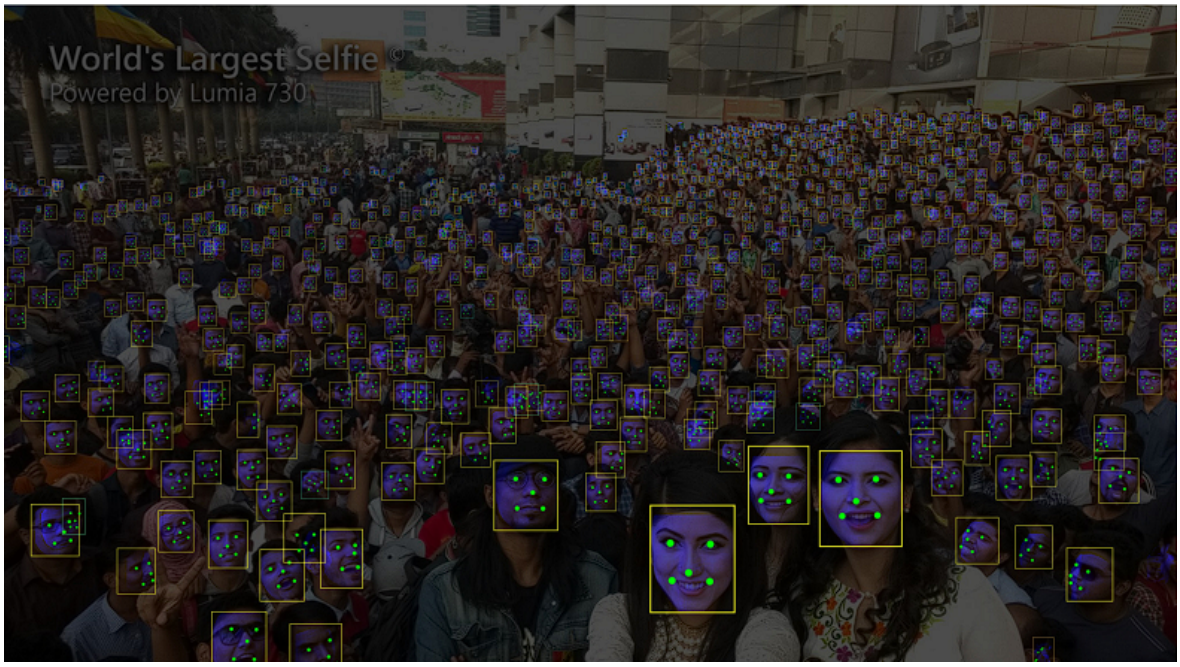

Figure 1.5: Example of RetinaFace library used on world's largest selfie [5].

We are using an implementation from [12] GitHub repository. You can see an example of this library's usage on the fig. 1.5.

1.4.4 6DRepNet

6DRepNet [6] is a state-of-the-art pre-trained PyTorch model for head pose estimation. It holds first place on a couple of popular head pose datasets and reliably gets to the

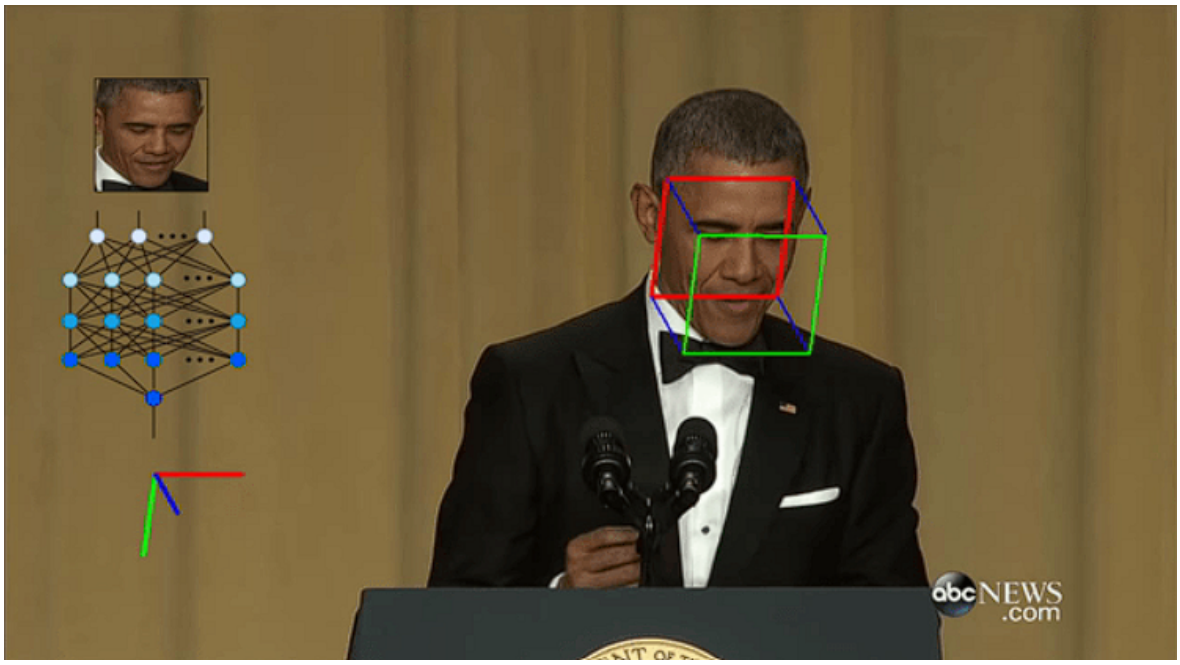

Figure 1.6: Example of 6DRepNet library used on a real image [6].

top 5 on others, as of writing this paper. It shows an advantage of up to 20% when compared to other state-of-the-art models.

The ability of 6DRepNet to learn and predict head positions beyond the narrow-angle constraints of most other models is one of its key advantages. The proposed network breaks this limitation by capturing the entire rotational view of the head, whereas previous approaches limited head pose estimation to a short range of angles to produce good results.

In our case, it was used for providing additional data to our main eye gaze estimation model for better accuracy. On the fig. 1.6 you can see it in action.

1.4.5 Unreal Engine and Metahumans

Epic Games created a powerful and popular game engine known as the Unreal Engine. It is known for its powerful ability to produce highly realistic and immersive experiences and has not only been used in traditional computer games but also in virtual reality, augmented reality, and even movie production. It is a popular choice for developers and designers who want to build realistic virtual environments due to its adaptability and easy-to-use interface. It also supports ray tracing for very realistic dynamic lighting, which is a key to real-looking computer graphics.

Unreal Engine's ability to interact easily with a wide array of technologies, allowing the production of lifelike characters and realistic animations, is one of its standout features. One such innovative technology is MetaHumans, an innovative suite of tools launched by Epic Games. With unprecedented detail and quality, MetaHumans enables

Figure 1.7: Example of characters generated with MetaHuman tool [7].

the creation of highly realistic and flexible digital human models. It offers realistic character creation tools, allowing developers to create lifelike virtual humans faster.

Developers can choose from a wide selection of ready-made digital humans with MetaHumans, representing a variety of ages, nationalities, and looks. These ready-made models can be easily adapted and customized, or even completely redone to meet specific project requirements. These characters look much more realistic due to MetaHumans' high level of realism, which includes facial emotions, hair, and clothing.

We are going to use these powerful tools for generating another dataset with varying head and eye positions, and lighting conditions. Examples of these realistic humans can be seen on fig. 1.7.

1.4.6 Blueprints

Blueprints, in the context of Unreal Engine, are a visual scripting system that allows users to develop interactive and dynamic behaviors for their games or applications without using conventional programming. By visually connecting nodes that represent various functions and actions, developers and designers can create intricate gameplay mechanics, user interfaces, and interactions using Blueprint.

By placing and connecting nodes in a graph-like setting, users of the Blueprints system can build and change game logic using an intuitive interface. Users can change the actions, functions, or conditions represented by each node by changing parameters and defining properties. The adaptability and flexibility of Blueprints is one of its main benefits. They allowed us to quickly and easily prototype and create "code" that

generates datasets without prior Unreal Engine experience. It has been an excellent tool for rapid prototyping and iteration since it doesn't have syntax errors, and when the blueprint is not that big, it is easy to read and understand.

In addition, blueprints provide a high degree of ecosystem connectivity with various components of the Unreal Engine platform. Characters and animations are some of the game assets that can be easily accessed and changed within the Blueprint editor. This made the task of manipulating Metahumans easy and intuitive. It did leave an impression, that if the code had to be bigger and more complicated it would not have been as good as traditional code, though, since the blocks of "code" that would represent each line of code anywhere else are quite a bit bigger on the screen and especially with the connections between them, they can take up a lot of screen space for a pretty basic program.

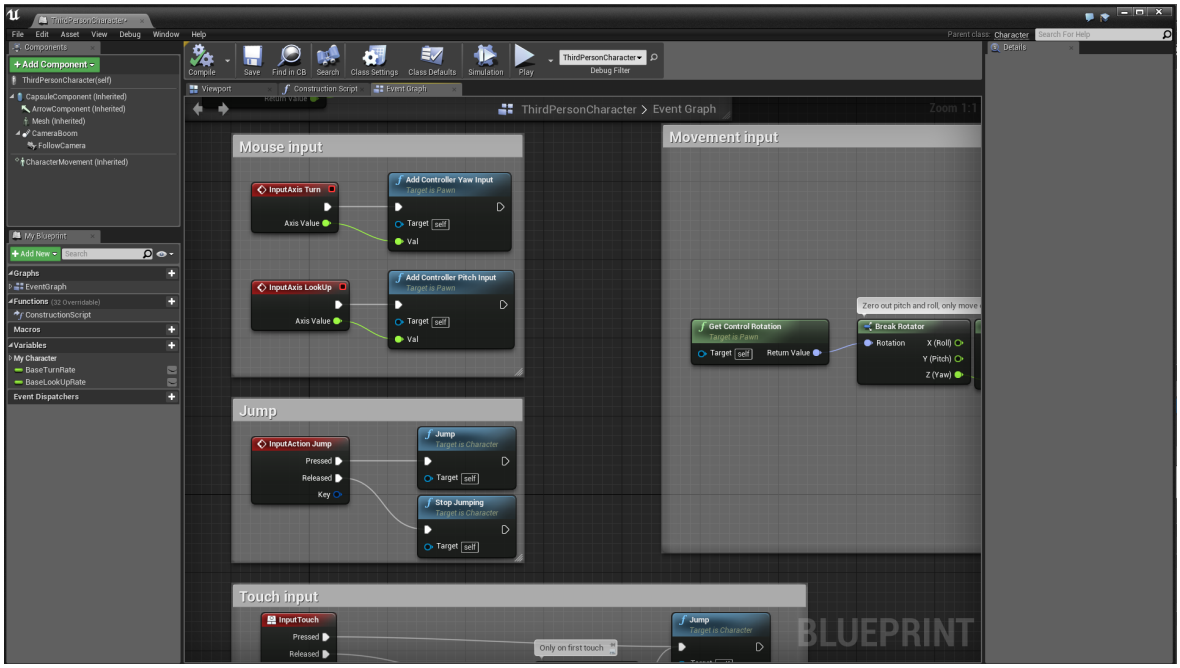

Figure 1.8: Blueprint editor screenshot [8].

On the fig. 1.8 you can see a blueprint editor window open with a basic character control blueprint. As you can see, even without any experience using them before, you can probably figure out what it does and how it does it. They also provide easy-to-understand and use debugging capabilities, which are always appreciated, especially since this was our first time working with them. Extensive documentation and community were also big help whenever we encountered any problems.

Chapter 2

Methodology

In this chapter, we will describe in detail the proposed eye gaze estimation system and the steps involved in its development. We will also describe the process of generating our own dataset using Metahuman technology.

2.1 Architecture

2.1.1 General architecture

To achieve precise eye gaze estimation in the context of human-robot interaction (HRI), the proposed eye gaze estimation system architecture is built by utilizing the capabilities of artificial neural networks (ANNs) and combining various components. The system architecture adopts a modular design strategy, including pre-trained models that are already in use and adding new components to address the problem of eye gaze estimation.

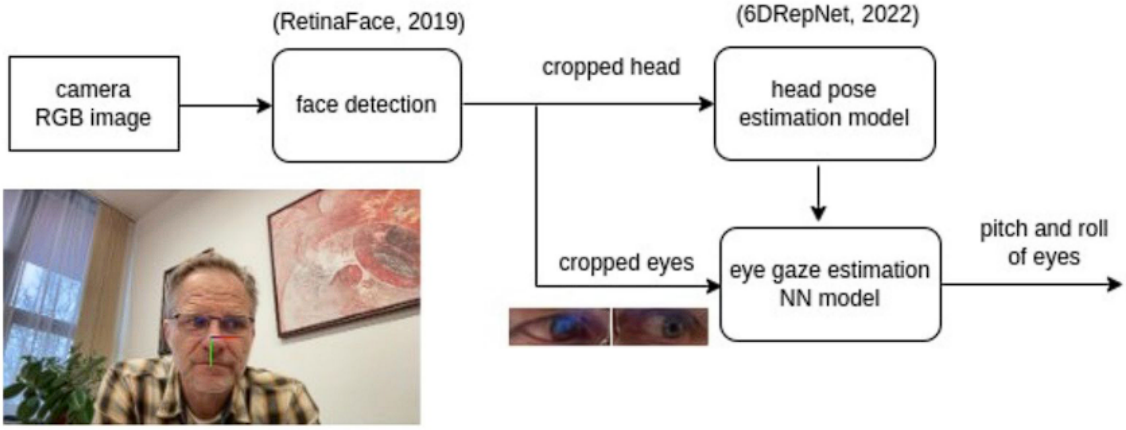

Figure 2.1: General architecture of eye gaze estimation system.

The overall layout of the architecture and several steps that make up the eye gaze estimation procedure are shown in fig. 2.1. Each step is thoroughly explained in the following subsections.

- Camera RGB image: Image is taken from webcam using OpenCV and rescaled to match input sizes for the models used.

- Face Segmentation: This step is for the separation of the human face from the source image. A pre-trained model, RetinaFace, is used to precisely locate and identify face regions. This step is essential for separating the face region for further processing since besides the bounding box of the face it also provides facial landmarks like positions of the eyes and nose. This data is then used to cut out the face and each eye.

- Head Pose Estimation: Using the pre-trained 6DRepNet model, the system calculates the pose or direction of the head in the second stage. The predicted data is used as an additional input to the eye gaze estimation model, which as we will show later is beneficial for its overall performance. Also, this data is used as an approximation of eye gaze when the head is at such an angle that the eyes cannot be seen properly. This is aimed at combating edge case scenarios since eye gaze estimation models outputs seem to be random when it can't see the eyes.

- Eye Gaze Estimation: The final step involves estimating eye gaze using a custom-built convolutional neural network (CNN) model. It takes two cropped eyes that are then concatenated together as shown on the fig. 2.1 as the input, and outputs two values in degrees. These are the predicted pitch and roll of the eyes.

2.1.2 CNN architecture

The basic ideas of the CNN architecture discussed in section 1.1 form the basis of our models architecture. You can see its architecture in fig. 2.2. To meet the unique needs and difficulties of better eye gaze estimation in the context of human-robot interaction, this architecture has undergone a number of enhancements and adaptations.

The amounts of parameters are not specified for each layer, since the model has been trained in a lot of different variations to get the best results. We will provide comparisons of these different models in the following chapters. In this architecture, the convolutional layers are responsible for figuring out different features (e.g. pupils, eyelids, edges of the eye, and so on) and then data about these features is processed by fully connected layers. We need to clarify that the mentioned features are just examples, and we don't know which exact features the model is using. Since convolutional layers work with 2D input, data is flattened before fully connected layers take it in, they use the information about those features and attempt to determine the gaze direction of the eyes and output the final values in degrees.

The estimated head position is provided only to the last fully connected layer since it has the smallest amount of inputs, so it is more probable to actually consider those

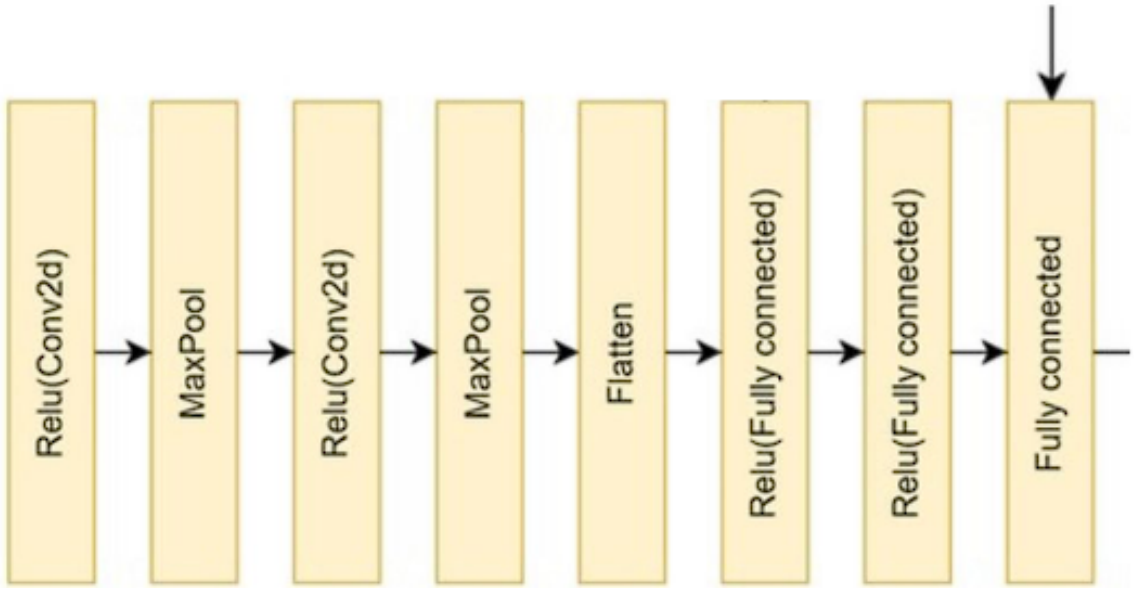

Figure 2.2: Architecture of the eye gaze estimation CNN.

values when calculating the final result.

2.2 Metahuman dataset

Metahuman tool for Unreal Engine was used to generate a dataset of over 57,000 images with 15 characters in different eye and head positions.

2.2.1 Characters

First, we used the Epic Games MetaHuman Creator program, which provides an easy-to-use interface for creating and customizing virtual humans. In our case, we did not customize any of them manually because there is already a database of premade humans, which was more than enough for us. That said, if for some reason we wanted to generate more data with more characters it would allow us to create a practically limitless amount of them with varying physical attributes like race, gender, age, eye color, and other physical characteristics. In our case, 15 diverse characters were chosen so that models trained on the dataset work equally well on everybody.

Also, it is important to point out that it would be possible to add attributes such as glasses to the characters, though it won't be as trivial since you will need third-party 3D modeling software and basic modeling skills.

Figure 2.3: The scene from Unreal Engine where the dataset was generated.

2.2.2 Setting up the scene

We used Unreal Engine, with its powerful real-time rendering engine, to create high-quality photos of MetaHumans in a virtual environment set up after the virtual human characters were created. You can see the scene created on fig. 2.3. Lighting conditions were changed throughout the generation process to make the models trained on it more resistant to those changes in the real world. We've changed the positioning of lights, their amount, and brightness. Real-time ray-traced lighting, provided by Unreal Engine 5 lumen technology, allowed us to do this dynamically without the need to bake lighting.

2.2.3 Generation blueprint

The generation algorithm for the photos is pretty basic. First, we generate a list of different values for both eye and head positions. Then we iterate over them and use the animation controller, that we've created, to change those positions and take a screenshot using the camera. This way, every character goes through 153 eye positions in each of the 25 head positions. This results in 3825 images generated per character. On the fig. 2.4 you can see a sample of the images generated this way.

This sort of generation can be expanded to use accessories such as glasses, different types of clothing, and different backgrounds. And it is not limited only to eye gaze estimation, it can be used to generate dataset visual datasets for everything that can be imagined. For example, it can be used to generate a dataset for head pose estimation or to generate cars on the street for driving automation. Basically, its possibilities are limitless and this is a lot cheaper and less time-consuming than creating such datasets in the real world. It also can generate more images than any real dataset could ever

Figure 2.4: Sample of pictures from the generated dataset.

have, and it doesn't need manual labeling, since the game engine already has all the positional and rotational data for all the objects in the scene.

Chapter 3

Implementation and testing

In this chapter, we will train, tune and test different eye gaze estimation models. We will also compare the results from some of these models on the Columbia Gaze Dataset and Metahuman Dataset. In doing so, we can evaluate the effectiveness of generating a dataset in real-world scenarios and evaluate its capacity to replace real datasets that are so much harder and more expensive to produce. We will also be able to pick the best model to use with a robot for real-life human-robot interaction.

3.1 Model iterations

3.1.1 Iterative improvements

The accuracy and robustness of the eye gaze estimation system were improved by a lot of tuning of hyperparameters and model architecture to achieve the final version. To evaluate iterative improvement, we've used a four-fold cross-validation on the Columbia Gaze Dataset. While for training the model mean squared error (MSE) was used to reduce deviation, for evaluating its performance on the testing data we've used mean absolute error (MAE). All the error values are averaged out between both output values (pitch and yaw). Also, it has to be noted that the input image is normalized so that RGB values are within the $[0, 1]$ range. The following is a breakdown of the key steps taken during the model refinement process:

- The neural network input was initially created by extracting the eyes from the photos and resizing their concatenated form to a uniform size of 300x900 pixels. With a maximum error of 20, the first neural network training resulted in a mean absolute error (MAE) of 3.4.
- Next, the crop size was dramatically changed to reflect the actual situation after it was discovered that the resolution of the cropped-out eyes from the photos taken by the robot camera was much lower than the photos on the data set. Now

the cropped-out eyes were 100x300 and this resulted in a significant reduction in model training and inference time. At the same time, the error metrics did not go up, so this did not affect the models' performance in testing. As for real-world testing, when using a lower-resolution webcam performance actually seemed to have improved, probably because crops from the camera were previously upscaled, to match the input size, while the new model was already working with these lower-quality images.

- A lot of experimentation was done with the amount and configuration of model layers. The configuration that you have seen on fig. 2.2 has been found to be the most optimal.

- To improve the performance of the model, the learning rate, batch size, activation functions, and dimensions of each layer were iteratively optimized. After all mentioned changes, we've reached an MAE of 2.6. (with a maximum error of 14) indicating an improvement in model accuracy.

- Another technique that we've tried was dynamic learning rate optimization when reaching the end of the model training process. But we've found no tangible effect since it was just trying to optimize the local peak performance instead of finding a new better peak.

- Different optimization algorithms were tried out including Adam, RMS Prop, SGD, and so on. Adam was found to be the best performer, so it was used from this point on.

- Another thing that has been really successful and improved the MAE by around 0.2 degrees was using the head pose, estimated by 6DRepNet, as an additional input into the last linear layer. At first, we tried to add this data to previous linear layers, but it did not have the desired effect. We think that the amount of input parameters was too big because of all the data from convolutional layers, and these values just got lost within all of it. But when moved to the last one, which has only around 50 inputs, it has shown great improvement in the model's accuracy. Also, subjectively, it seemed like the model was more robust when turning the head in different directions.

  Interestingly, when we tried to normalize the data to be within the $[-1, 1]$ range the positive change disappeared, so in this case having normalized deteriorated the performance, defying the common expectation.

- To reduce models' training time and find the best compromise between accuracy and computational efficiency, we've trained a couple more models with different

input image dimensions. The lowest one we've tried was 40x120, but the MAE did not go below 3.5, so after some experimentation, we've found 70x210 resolution to be the most optimal. At this point, we've reached an MAE of 2.1 and a maximum value of 11.

- We've also tried to turn images into black and white, hoping to improve performance in different lighting conditions, since the images would have been more similar. But it actually decreased the accuracy of the model and when trying it out in the real world no improvement was found when trying out different lighting scenarios.

- The last improvement to the MAE we were able to get was from expanding the data, which is not really model optimization but is still important to acknowledge. And we are not talking about generating datasets yet, this is actually a lot more basic data augmentation technique since all we did was mirroring all the images from the initial dataset. This works because our faces are not symmetrical, and the lighting conditions are also not perfect, so the images are a little different. Also, the segmentation of the eyes did not just return mirrored copies of eye images since it is a machine learning model and its performance is not symmetrical as well. This way, we were able to double the dataset size and improve our models' performance to around 1.9 MAE. The maximum error did not change, though.

3.1.2 Final model

After all the improvements described earlier, we've arrived at an architecture that you can see on table 3.1.

Layer (type)	Output Shape	Parameter #
Conv2d-1	[-1, 9, 68, 208]	252
MaxPool2d-2	[-1, 9, 22, 69]	0
Conv2d-3	[-1, 26, 20, 67]	2,132
MaxPool2d-4	[-1, 26, 6, 22]	0
Linear-5	[-1, 600]	2,059,800
Linear-6	[-1, 53]	30,050
Linear-7	[-1, 2]	108

Table 3.1: Summary of the network architecture

In total, the model has 2,092,342 parameters, and all of them are trainable.

3.2 Dataset testing and comparison

We conducted our testing on three separate datasets: the Columbia dataset, the Metahuman dataset, and a combination of both to assess the performance and generalizability of our approach. To avoid overfitting, we also used a comparison of validation loss against training loss.

3.2.1 Fighting overfitting

We use a technique based on comparing validation loss against training loss on a graph to avoid overfitting and ensure the capacity of the model to generalize. We track the loss values on the training and validation sets during the training process and plot them against the number of epochs. You can see the graph for these losses for Metahuman Dataset and Columbia Gaze Dataset on the fig. 3.1 and on the fig. 3.2 respectively

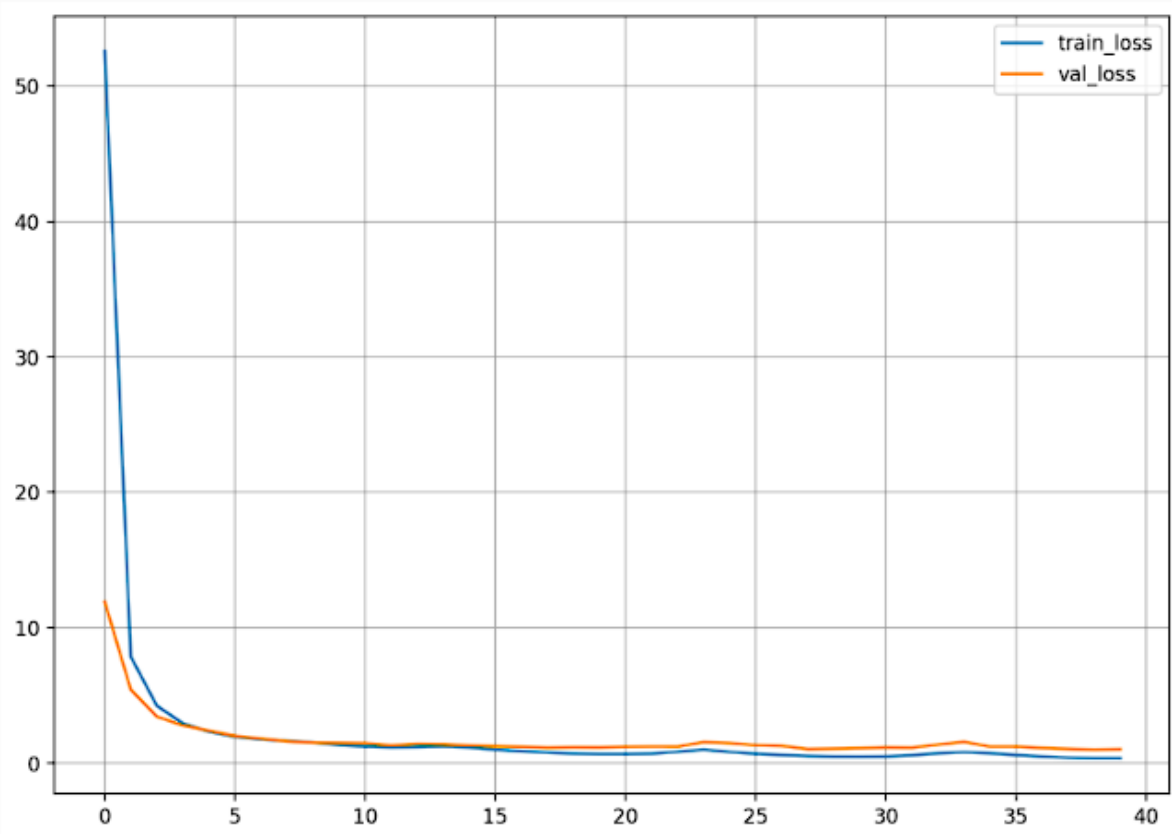

Figure 3.1: Training and validation loss comparison graph for the Metahuman Dataset

We can determine the time at which the model's performance on the validation set starts to degrade, compared to the training loss, or flatten out, indicating a potential overfitting scenario, by looking at the loss pattern on the loss graphs. We can find the best time to stop the training process using this data as a guide. Values extracted from these graphs were used for all future comparisons.

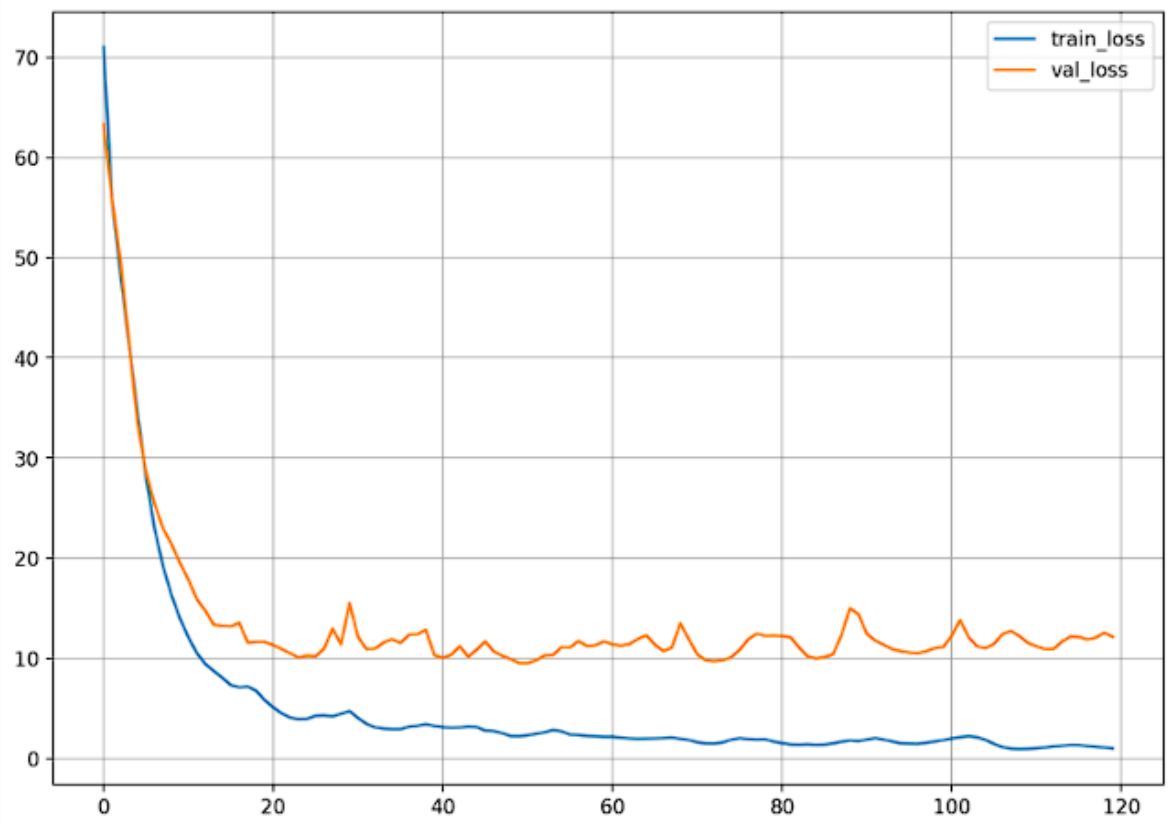

Figure 3.2: Training and validation loss comparison graph for the Columbia Gaze Dataset

3.2.2 Combined dataset

We integrated the Columbia Gaze and Metahuman datasets into one to improve the generalization ability of the model. With the combination of real-world and generated data, a more diverse and varied training set is expected. We aim to utilize the advantages of both datasets and improve the pose estimation performance over a wider range of parameters like lighting scenarios, head poses, and wearing or not wearing glasses.

3.2.3 Dataset comparison

After running four-fold cross-validation across Columbia Gaze Dataset, Metahuman Dataset and the combined dataset, we've got a result that you can see at fig. 3.3

As you can see, the most stable model across all three datasets is the one trained on the combined dataset, which is expected, but what this means is that by combining the two datasets we get the most diverse and generalized model. This model is the one we've chosen to be the main model that will be published as the pre-trained version.

If we just look at the results of the model when only trained on the Columbia Gaze Dataset, we can compare it to other models that were trained in other papers. For example, in one of the best models presented in [11] paper, their model achieves an

Trained on \ Tested on	Metahuman Dataset	Columbia Gaze Dataset	Combined Dataset
Metahuman Dataset	MAE ≈ 0.59 σ ≈ 0.52	MAE ≈ 8.12 σ ≈ 5.81	MAE ≈ 1.9 σ ≈ 3.79
Columbia Gaze Dataset	MAE ≈ 8.42 σ ≈ 4.83	MAE ≈ 1.93 σ ≈ 1.5	MAE ≈ 7.55 σ ≈ 5.17
Combined Dataset	MAE ≈ 0.65 σ ≈ 0.56	MAE ≈ 2.88 σ ≈ 1.94	MAE ≈ 1.04 σ ≈ 1.25

Figure 3.3: Comparison matrix of cross-validated results between the three datasets

MAE of around 3.8 on the Columbia Gaze Dataset across five-fold cross-validation. We outperform this result, almost cutting the loss in half. But we have to mention that the model has a problem with stability in harder lighting scenarios, and that is exactly why the model trained on the combined dataset is superior.

3.3 Real world testing

It was also very important to evaluate the model in real-world scenarios. In our research, we used a computer webcam and a NICO [13] robot to conduct real-world testing. As a result, we were able to subjectively evaluate the accuracy and robustness of the model in real-world situations and confirm that it can be later used in the context of human-robot interaction.

While our testing did not include a wide variety of people, we are sure that it would perform just as well on everybody else, since the datasets used were very diverse, and testing on said datasets has shown very high accuracy. Throughout the real-world testing, we've used the model under varied environmental circumstances, such as various lighting settings and backgrounds. You can see example photos of the model predicting eye gaze direction in different head positions and light conditions on the fig. 3.4, the blue line represents the predicted eye direction. Our subjective opinion is that the model trained on the combined dataset performed better in worse lighting scenarios than the one trained only on the Columbia Gaze Dataset, so we think that adding generated images to improve the performance of the model in those scenarios was a success.

In addition, we tried the eye gaze estimation model with the NICO humanoid robot, a platform created to study human-robot interaction. As a result, we were able to assess the effectiveness of the model using the inbuilt eye camera. The model performed pretty well in this type of testing, and we think that performance can be improved even further by calibrating a lens correcting algorithm on the camera since its lens is fish-eye. This would make the geometry on the pictures closer to reality and probably

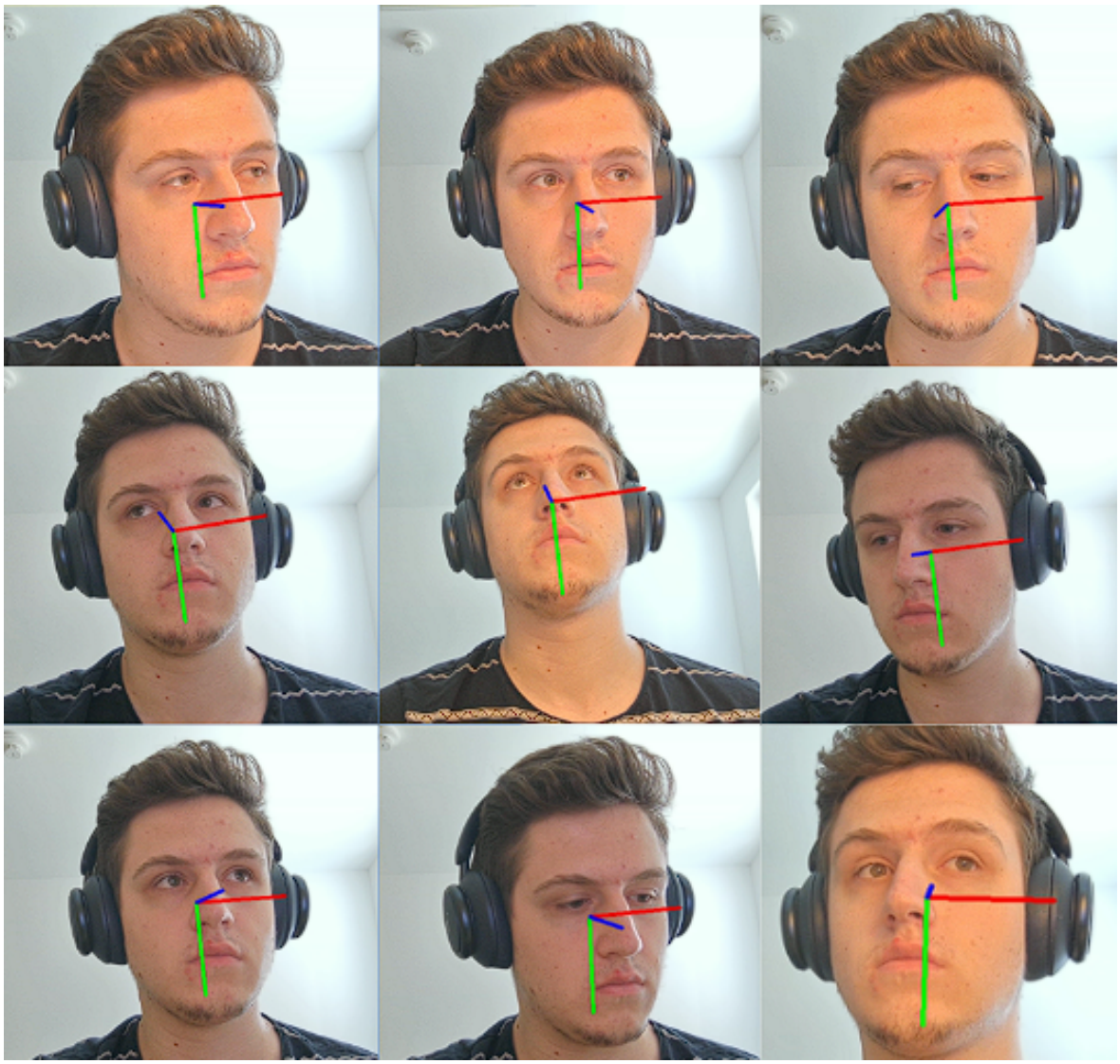

Figure 3.4: Example images from real-world testing with a webcam.

make the gaze estimation process easier for the model. You can see examples of the model predicting eye direction on fig. 3.5.

Also, after testing out the model that was only trained on the Metahuman dataset in the real world, we've found it to successfully predict the eye gaze of real people, so we think that generating datasets with modern game engine technologies like Unreal Engine has a promising future. Especially, since it allows creating a dataset of visual or 3D data for a wide variety of tasks at significantly lower costs and time compared to doing so in the real world. The code, pre-trained model, and the Metahuman project will be published and available through a GitHub repository for anybody else to use for their own research. Also, the model is planned to be used for further human-robot interaction research at our university.

Figure 3.5: Example images from real-world testing with the NICO robot.

Conclusion

In this thesis, we explore the exciting field of eye gaze estimation for human-robot interaction, focusing on the important contribution of estimation to improving the effectiveness of these interactions. We are leveraging the capabilities of Artificial Neural Networks (ANNs) to address this problem, as we recognize the inherent difficulties in the field of eye gaze estimation caused by significant variability in eye gaze direction and head location.

Investigating current eye and head position estimation models and developing a reliable eye gaze estimation system using various pre-trained models, and our own model, was our primary goal. This technology is intended to work with a wide range of applications, including robots. The flexibility of our suggested solution, to operate with standard RGB cameras without the need for additional hardware or IR filters, is one of its distinctive features. The system was created to function effectively in varying head and eye positions, with the camera position independent of the head position.

Using the potential of Unreal Engine and Metahumans tools in the creation of diverse and high-quality datasets was really beneficial to our research. Using this technology, we can create customized datasets that take into account different head and eye orientations and lighting conditions. These datasets can be a very useful resource for further research in this area, to improve the performance of our models. There is also potential to leverage the cutting-edge capabilities of Unreal Engine to generate more complicated data sets that are more diverse, have harder lighting scenarios, and have characters wearing glasses, possibly creating opportunities for the creation of more sophisticated and precise eye gaze estimation models.

After combining some pre-trained models with our custom one, we see encouraging findings, with our system showing great results predicting eye gaze on different datasets and in the real world. This supports our hypothesis about ANN's potential for eye gaze estimation tasks. While the results are encouraging, we realize that there is always room for improvement with both expanding the dataset and optimizing the model, for better results in bad light conditions, or for example for people wearing glasses. Another weak point, that any system using a standard RGB camera will have, is problems with predicting eye gaze directions when a person is looking down and the camera is positioned above their eye level. This happens because eyelids are almost completely

obscuring pupils. The easiest way to fix this is just to have the camera below the person's eye level, which is pretty common for robots.

In conclusion, the presented pre-trained models and generated dataset will be available and will probably be used for further research in the field of human-robot interaction. The code and all other resources are accessible through the GitHub repository.

Bibliography

[1] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[2] B.A. Smith, Q. Yin, S.K. Feiner, and S.K. Nayar. Gaze locking: Passive eye contact detection for human-object interaction. In *ACM Symposium on User Interface Software and Technology*, 2013.

[3] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Appearance-based gaze estimation in the wild. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4511–4520, June 2015.

[4] Erroll Wood, Tadas Baltrušaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. Learning an appearance-based gaze estimator from one million synthesised images. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research Applications*, pages 131–138, 2016.

[5] Jiankang Deng et al. RetinaFace: Single-stage dense face localisation in the wild. In *arXiv:1905.00641v2*, 2019.

[6] Thorsten Hempel, Ahmed A. Abdelrahman, and Ayoub Al-Hamadi. 6D rotation representation for unconstrained head pose estimation. In *2022 IEEE International Conference on Image Processing*, 2022.

[7] MetaHuman high-fidelity digital humans made easy. `https://www.unrealengine.com/en-US/metahuman`, 2021.

[8] Introduction to blueprints. `https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/`.

[9] Oskar Palinko, Francesco Rea, Giulio Sandini, and Alessandra Sciutti. Robot reading human gaze: Why eye tracking is better than head tracking for human-robot collaboration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5048–5054, 2016.

[10] Jaekwang Oh, Youngkeun Lee, Jisang Yoo, and Soonchul Kwon. Improved feature-based gaze estimation using self-attention module and synthetic eye images. *Sensors*, 22(11), 2022.

[11] Seonwook Park, Adrian Spurr, and Otmar Hilliges. Deep pictorial gaze estimation. In *European Conference on Computer Vision*, pages 741–757, 2018.

[12] Face detection. `https://github.com/elliottzheng/face-detection`.

[13] Matthias Kerzel et al. NICO – Neuro-Inspired COmpanion: A developmental humanoid robot platform for multimodal interaction. In *IEEE International Symposium on Robot and Human Interactive Communication*, 2017.