TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Deep Learning-based AR Glasses and Head Tracking for Automotive Augmented Reality

Thesis approved by the
Department of Computer Science
Technische Universität Kaiserslautern
for the award of the Doctoral Degree
Doctor of Engineering (Dr.-Ing.)
to

Ahmet Firintepe

Date of Defense:	12. September 2022
Reviewer:	Prof. Dr. Didier Stricker
Reviewer:	Prof. Dr. Gudrun Klinker
Dekan:	Prof. Dr. Jens Schmitt

DE-386

Abstract

In recent years, Augmented Reality has made its way into everyday devices. Most smartphones are AR-enabled, providing applications like pedestrian navigation, Point of Interest highlighting, gaming, and retail. The high-tech industry has been focused on developing smartglasses to present virtual elements directly in front of the viewers' eyes, allowing more immersive AR experiences. Smartglasses can also be deployed while driving for an enhanced and more safe experience. A 3D registered augmentation of the real world with navigation arrows, lane highlighting, or warnings can decrease the duration of inattentiveness regarding driving due to glancing at other screens. Enabling HMDs' usage inside cars requires knowing its exact position and orientation (6-DoF pose) in the car. This necessitates sensors either built inside the AR glasses or the car. In a car, the latter option called outside-in tracking is more attractive due to two reasons. First, AR glasses containing different sensor sets exist, hampering finding one single solution for different HMDs. Second, the view from the driver's perspective combines static interior and dynamic exterior features, complicating finding a reliable set of features. Nowadays, tracking methods utilize Deep Learning for a more generalizable and accurate derivation of the 6-DoF pose. They achieve outstanding results for head and object pose estimation.

In this thesis, we present Deep Learning-based in-car 6-DoF AR glasses pose estimation approaches. The goal of the work is an exploration of accurate HMD pose estimation with the help of neural networks. The thesis achieves this by investigating numerous pose estimation techniques. Evaluations on the recorded HMDPose dataset constitute the foundation for this, consisting of infrared images of drivers wearing different HMD models. First, algorithms based on images are derived and evaluated on the dataset. For comparison, we carried out an evaluation on image-based methods considering time information. Further, pose estimation based on point clouds, generated out of infrared images, are analyzed. An investigation of various head pose estimation methods to derive its potential use are conducted.

In conclusion, we introduce several highly accurate AR glasses pose estimators. The HMD pose alone achieves better results than the head pose and the combination of the head and HMD. Especially our image-based methods with optional usage of time information can efficiently and accurately regress the AR glasses pose. Our algorithms show excellent estimation results on live data when deployed inside a car, making seamless in-car HMD usage possible in the future.

Acknowledgment

First of all, I would like to thank my PhD supervisors Dr. Jörg Preißinger at BMW and Prof. Dr. Didier Stricker at the University of Kaiserslautern for giving me the opportunity to work on this thesis. The contribution I was able to make to the science community and industrial advancement at the same time was very important and meaningful to me. Thank you Dr. Alain Pagani for supervising my thesis and giving me valuable and regular advice and sharing your experiences with me. Further I would like to thank my initial second advisor Prof. Dr. Heinrich Hußmann for his support, may he rest in peace. Thank you Prof. Dr. Gudrun Klinker for stepping in as my second advisor and accepting the judging of my thesis.

I also want to thank all my colleagues in the Augmented Reality Glasses project at BMW. Special thanks to Dr. Alan Keim and Dr. Wolfgang Haberl for inspiring discussions and help on continuously progressing on my topic. I would also like to thank the many students who helped me during implementation and testing of the various algorithms of this thesis.

Finally, I would like to thank my family and my friends who supported and motivated me over the years to achieve my goals.

Contents

1 Introduction **11**
- 1.1 Motivation 11
- 1.2 Problem Formulation and Approach 12
- 1.3 Contributions 14
- 1.4 Organization of the Thesis 15
- 1.5 Publications 16

2 Background **17**
- 2.1 Augmented Reality 17
  - 2.1.1 Definition and Differentiation 17
  - 2.1.2 In-Car AR Systems 20
- 2.2 Machine Learning and Deep Learning 22
  - 2.2.1 Machine Learning 22
  - 2.2.2 Deep Artificial Neural Networks 23
  - 2.2.3 Deep Learning in Computer Vision 26
- 2.3 Pose Estimation and Tracking 28
  - 2.3.1 Deep Learning vs. Feature-based Pose Estimation 29
  - 2.3.2 Head Pose Estimation 30
  - 2.3.3 Object Pose Estimation 30
  - 2.3.4 Inside-Out and Outside-In Pose Estimation 31
  - 2.3.5 Pose Estimation Algorithms in the Automotive Context 32

3 Error Ranges for Augmented Reality Car Use Cases **33**
- 3.1 Errors in Augmented Reality Systems 33
  - 3.1.1 Acquisition Error 33
  - 3.1.2 Head-Tracking Error 34
  - 3.1.3 Display Error 35
  - 3.1.4 Viewing Error 35
  - 3.1.5 Relevant Errors for AR Systems in Car Setups 36
- 3.2 Error Ranges of Car Use Cases 36
  - 3.2.1 Car Interior 38
  - 3.2.2 Car Environment 39
  - 3.2.3 Distant Objects 39
- 3.3 Summary 40

4 Deep Learning-based AR Glasses Pose Estimation 41
- 4.1 Image-based Feed-Forward Pose Estimation . . . 42
  - 4.1.1 GlassPose Network . . . 42
  - 4.1.2 GlassPoseRN Network . . . 43
  - 4.1.3 CapsPose Network . . . 44
  - 4.1.4 Benchmarked State-of-the-Art Image-based Method . . . 48
- 4.2 Image-based Recurrent Pose Tracking . . . 49
  - 4.2.1 BaselineCNN - Backbone for RNN methods . . . 49
  - 4.2.2 LSTM-based Approaches . . . 49
  - 4.2.3 RNNs with Non-Local Blocks . . . 51
  - 4.2.4 Sequential CapsPose Network . . . 53
- 4.3 Point Cloud-based Pose Estimation . . . 55
  - 4.3.1 Point Clouds Generation . . . 55
  - 4.3.2 Pose Representation . . . 60
  - 4.3.3 PointsToRotation Network . . . 60
  - 4.3.4 PointsToPose Network . . . 62
  - 4.3.5 Benchmarked State-of-the-Art Point Cloud-based Method . . . 64
- 4.4 Neural Network Ensemble Methods . . . 64
  - 4.4.1 Shallow Fusion Methods . . . 64
  - 4.4.2 Learning-based Deep Fusion . . . 66
- 4.5 Summary . . . 69

5 Deep Learning-based Head Pose Estimation 71
- 5.1 In-Car Head Pose Estimation on the AutoPOSE Dataset . . . 71
  - 5.1.1 Dataset Preparation . . . 71
  - 5.1.2 Neural Networks for Head Orientation Estimation . . . 72
- 5.2 Adaptation of AR Glasses Pose Techniques for Head Pose . . . 76
  - 5.2.1 Image-based Feed-Forward Head Pose Estimation . . . 76
  - 5.2.2 Image-based Recurrent Pose Tracking . . . 78
  - 5.2.3 Point Cloud-based Pose Estimation . . . 79
- 5.3 Fusion of HMD and Head Pose . . . 80
- 5.4 Summary . . . 83

6 Large-Scale 6-DoF HMD and Head Pose Dataset 85
- 6.1 HMDPose - an AR Glasses Dataset . . . 85
  - 6.1.1 Hardware Setup . . . 85
  - 6.1.2 Calibration and Synchronization . . . 86
  - 6.1.3 Acquisition and Dataset Analysis . . . 87
  - 6.1.4 Postprocessing . . . 89
- 6.2 Head Pose Annotation Generation on the HMDPose Dataset . . . 90
  - 6.2.1 Facial Landmarks Extraction & Landmark Reconstruction . . . 91
  - 6.2.2 Head Pose Computation and Postprocessing . . . 92
  - 6.2.3 Comparison of Head Pose and AR Glasses Pose Labels . . . 93

6.2.4 Data Analysis 96
6.3 Summary 98

7 Evaluation of Head and AR Glasses Pose Estimation 99
7.1 Evaluation Metrics 99
7.2 Head Orientation Estimation Evaluation on the AutoPOSE Dataset 100
7.3 AR Glasses Pose Estimation Evaluation 104
7.3.1 Feed-Forward Pose Estimation Results 104
7.3.2 Recurrent Pose Tracking Results 109
7.3.3 Point Cloud-based Pose Estimation Results 115
7.4 Head and AR Glasses Pose Estimation Comparison 119
7.4.1 Data Split Strategies 119
7.4.2 Head and HMD Comparison Results 119
7.4.3 Generalization Analysis 124
7.5 Evaluation of Fusion Methods 126
7.5.1 Ensemble Selection 126
7.5.2 Neural Network Ensemble Results 127
7.6 Evaluation in Real-World Conditions 130
7.6.1 Neural Network Comparison 131
7.6.2 Evaluation on Unknown Glasses 135
7.7 Discussion 137
7.7.1 Pose Estimation Performance 137
7.7.2 Memory Requirement & Inference Time 138
7.8 Summary 139

8 Conclusion 141
8.1 Summary 141
8.2 Future Work 142

List of Figures 143

List of Tables 144

Bibliography 146

1 Introduction

The objective of this thesis is the study of pose estimation and tracking methods of Augmented Reality (AR) glasses and head inside the car using Deep Learning. This has the purpose of enabling accurate tracking of AR glasses during driving. We use ”AR glasses”, ”glasses”, ”smart-glasses”, ”data glasses” and ”Head-Mounted Display (HMD)” interchangeably in this thesis, whereas HMD always refers to AR-enabled HMDs. We achieve a broad study on AR glasses tracking with different inputs and different neural network architectures. Hence, questions for the goal of HMD pose estimation arise.

First, there are questions requiring answers about the input type for the neural networks: Are grayscale infrared (IR) images sufficient for an accurate pose estimation? Do point clouds generated from infrared images enhance the pose performance?

Then, questions on the neural network architecture need to be answered: Do multi-view images improve the estimation accuracy, or are single images performant enough? Do recurrent models utilizing the time information of image sequences improve the estimation accuracy? How deep do neural networks have to be built for a low error pose estimation?

The last question category points towards head pose estimation: How does head pose estimation compare to AR glasses pose estimation? Can it substitute the HMD pose or improve its performance when combined?

In this work, we will answer these questions by studying different input types like one or more IR images and point clouds for numerous neural networks based on various architectures for AR glasses and head pose estimation.

1.1 Motivation

After decades of work on Augmented Reality in the research community, its first deployment in the industry was conducted by Boeing in 1990. An head-mounted display was proposed for construction workers to project instructions on the wiring of certain cables in aircrafts. Instead of using different boards for each plane with instructions about the wiring, they were shown digitally through the HMD. Another early adoption of AR was also conducted in the aerospace industry with the usage of head-up displays for pilots. They are still used to show information to the pilot relevant to maneuver the plane in their direct field of view.

Since then, constant progress in industry and research has been made and Augmented Reality (AR) is currently on its way into our daily lives. AR applications and AR-dedicated hardware have become part of most smartphones. Further, many companies have started to commercialize AR glasses (Figure 1.1). Already available glasses like Microsoft Hololens have shown the capabilities of AR thanks to extensive usage of sensors. A variety of built-in sensors enable

(a) Google Glass [1]. (b) Microsoft Hololens 2 [2].

Figure 1.1: Two widely known Augmented Reality Glasses.

highly accurate tracking, which is crucial for seamless and accurate display of AR content. In general, in order to ensure a convincing experience, it is paramount to localize the used smartphone or AR glasses with high accuracy in its environment. Usually, this is required in a highly accurate and latency-free way. Smartphones are more error-tolerant regarding tracking, as they are equipped with video-see-through displays. They have the advantage of frequency and speed adjustment of the real world shown through the display. AR glasses steadily matured over the years and are currently being used in the industry. In the future, consumer adaptation of AR glasses is highly likely. In contrast to smartphones, AR glasses mostly consist of optical-see-through displays, where the wearer perceives the real world directly. This elevates the latency and accuracy requirements for a realistic experience.

Most AR glasses track their environment based on built-in sensors - typically cameras, depth-cameras, and inertial measurement units (IMUs) - to later superimpose virtual content on top of real-world elements. This is sensible for static environments and in the case of reliance on one particular vendor's glasses model with a specific sensor set. In dynamic environments like airplanes or cars, IMUs register both head and vehicle movement, making separation difficult. Thus, it is relatively challenging to utilize them in a standalone manner while driving. In the case of a car driver, the built-in optical sensors register mostly parts of the outside world, which delivers a limited set of features of the car interior for inside-out tracking.

In a dynamic platform like a car with a goal of enabling varying types of glasses, outside-in tracking can help. In this case, cameras are pointed at the driver to track the worn glasses. Algorithms adjusted to handle different types of glasses models can support scalability to new types of models. Thus, outside-in tracking can lead to a wider adoption of AR glasses while driving.

1.2 Problem Formulation and Approach

Augmented Reality is a powerful and intuitive technology that enables a variety of use cases when deployed inside cars. Especially the fully usable area inside and outside the car as AR glasses move with the head movement make them particularly interesting. However, despite their capability of supporting an abundance of useful use cases, to the best of our knowledge, estimating their position and orientation based on cameras has not been thoroughly studied. The position and orientation, which constitute the 6-DoF pose of the HMD, are essential for AR. In the car, there are many use cases that are projected in a far distance. This necessitates a highly accurate pose to support seamless AR immersion.

Although there have been works in the field of object and head pose estimation, they were not conducted for the purpose of enabling AR content in HMDs. Works on object pose estimation usually focus on RGB or depth sensor-based algorithms. In the car use case, infrared cameras are helpful due to their invariance to illumination changes, which has made them the state-of-the-art for cars. Head pose estimation methods have previously been developed with infrared images as input. Nevertheless, they are not directly transferable, as there are no HMDs on the images, including many valuable facial features. Moreover, the visibility of the HMD and the head at all times constitutes a unique challenge, which has not been addressed yet. The object and head have been targeted separately.

This thesis aims to develop and study highly accurate AR glasses and head pose estimation algorithms based on infrared images. To quantify this, we define various error ranges for AR car use case clusters depending on the projection distance of the content. In each cluster, different use cases require more or less accurate poses to work correctly. Our goal is to investigate how different types of neural networks or input types affect AR glasses pose estimation and how head pose estimation compares to it. In addition, we want to analyze the potential of combining several pose estimation algorithms, also regarding head and AR glasses pose.

Approaches

The approaches to be investigated can be divided into four groups: networks based on one or more IR images as input, networks deriving point clouds as an intermediate presentation for estimating the pose, algorithms that specifically incorporate the time information into the network architecture, and fusion methods.

Image-based Feed-Forward Pose Estimation encapsulates AR glasses and head pose estimation methods based on only one or more IR images. We investigate several Deep Neural Networks. We work with various architecture types, ranging from deeper ResNet-based [3] architectures to networks with fewer layers to Capsule-based [4] architecture types.

Point Cloud-based Feed-Forward Pose Estimation includes pose estimation algorithms, which use point clouds as an intermediate representation. We develop a semi-supervised point cloud estimation approach, where the output is used by multiple neural network types that we designed. The pose estimators contain the well-established PointNet architecture [5, 6], which selects the most suitable points in the input for the task at hand.

Image-based Recurrent Pose Tracking presents image-based pose tracking algorithms. The algorithms utilize LSTM-layers [7] to make use of the temporal information present in continuous data.

Fusion Methods comprise several fusion methods to evaluate if fusing several pose estimation algorithms can enhance the overall performance. This includes AR glasses and head pose trackers. We propose fusion algorithms that either rely on the output of the pose estimation networks directly without further learning or add shared layers to the networks and train them.

1.3 Contributions

The main contribution of this thesis is the systematic study of in-car outside-in AR glasses pose estimation. We define three categories of estimating the AR glasses pose based on infrared images. In the analysis of the 6-DoF AR glasses pose estimation problem, we develop many different technical solutions per category. We further investigate head pose estimation in this context. Table 1.1 shows the introduced algorithms per category, their input type, if they were trained on head or glasses pose, and the number of images they were trained with.

Category	Approach	Input Type	Head or Glasses	# Images
Image-based Feed-Forward Pose Estimation	GlassPose	IR Images	Glasses	1-3
	GlassPoseRN	IR Images	Both	1-3
	CapsPose	IR Images	Both	3
	HON	IR Images	Head	1
	ResNetHG	IR Images	Head	1
Image-based Recurrent Pose Tracking	CNN-LSTM	IR Images	Glasses	3
	NL-CNN-LSTM	IR Images	Glasses	3
	SepConv-LSTM	IR Images	Both	3
	NL-SepConv-LSTM	IR Images	Glasses	3
	GlassPoseRN-LSTM	IR Images	Glasses	3
	SeqCapsPose	IR Images	Both	3
Point Cloud-based Feed-Forward Pose Estimation	P2R	Point Cloud	Glasses	-
	P2P	Point Cloud	Both	-

Table 1.1: An overview of our novel algorithms and some of their properties.

The technical contributions are the following:

- An AR glasses pose dataset called HMDPose. It includes infrared triple images of four different AR Head-mounted displays captured in a car. It contains sequences of 14 subjects captured by three different cameras running at 60 FPS each, adding up to more than 3,000,000 labeled images in total. It includes ground truth 6-DoF-poses, captured by a submillimeter accurate marker-based tracker. It is publicly available on ags.cs.uni-kl.de/datasets/hmdpose/ [8].
- Three custom-developed methods for image-based feed-forward pose estimation. We compare the network performances in a study when using three infrared images as input. Especially our GlassPoseRN and CapsPose methods show high accuracies. In addition, an evaluation with two of the novel methods is conducted on single and multi-view pose estimation. Both networks were trained and tested with one, two, and three input images on the HMDPose dataset. The networks show accurate pose estimation results independent of the number of inputs and show some improvements in position estimation with more input views [9].

- Six outside-in approaches on recurrent pose tracking of AR glasses. In our evaluation, two frame-by-frame pose estimation approaches, including the mentioned GlassPoseRN method, are compared to six LSTM-based recurrent methods with a CNN backbone, incorporating the time information of image sequences. We compare methods with and without non-local blocks or separable convolutions. The recurrent tracking methods show improved results compared to frame-by-frame pose estimators [10].

- Two AR glasses pose estimation algorithms, working on 3D point clouds as an intermediate representation. A point cloud estimator is introduced, trained on multi-view infrared images without depth ground truth in a semi-supervised manner, generating point clouds based on one image only. The first pose estimation approach ”PointsToRotation” is based on a Deep Neural Network alone. The second approach ”PointsToPose” combines Deep Learning and a voting-based mechanism. Both networks show accurate results [11].

- A study with two novel networks on in-car head pose estimation on IR images of the AutoPOSE dataset. The introduced networks are named Head Orientation Network (HON) and ResNetHG have been benchmarked on different input resolutions. In addition, multiple depths have been evaluated within the HON and ResNetHG networks regarding their effect on accuracy [12, 13].

- A comparison of AR glasses and head pose estimation performance. We adapt various AR glasses pose estimation approaches and train them on head pose labels generated off the HMDPose dataset. We show that estimating the AR glasses pose is more accurate than the head pose in general. Fusing AR glasses and head brings limited improvement concerning the overall pose. We also analyze the networks’ performance, in case either all AR glasses or all faces are known to the networks during training. Thus, the performance regarding driver or AR glasses generalization has been investigated [14].

1.4 Organization of the Thesis

This thesis is organised in the following way:

Chapter 2 introduces background information on Augmented Reality in the car context, Machine Learning and Deep Learning, and pose estimation.

Chapter 3 is devoted to the definition of error ranges of pose estimation for Augmented Reality. After a review of potential error sources in an AR system, we introduce AR use case categories in the car context depending on their projection distance. We then define pose accuracies required per use case for an immersive superimposition.

Chapter 4 presents various Deep Learning-based AR glasses pose estimation approaches. Several methods per pose estimation category are described in detail. Fusion approaches to combine different pose estimators finalize the chapter.

Chapter 5 addresses Deep Learning-based head pose estimation. An overview with various head pose estimation methods on the AutoPOSE dataset is given first, which later served as a foundation for our final AR glasses and head pose methods. Afterwards, the final head pose estimation networks are presented.

Chapter 6 is dedicated for the evaluation of the previously defined algorithms. Most importantly, the chapter evaluates the three different categories of AR glasses pose estimation and their algorithms. Then, head and AR glasses pose performance is compared in detail. An evaluation of fusing the networks and a real-world study round off the chapter.

Finally, Chapter 7 concludes the thesis and identifies directions for future work.

1.5 Publications

Most of the work presented in this thesis has been accepted and presented in peer-reviewed conferences or journals. In the following, we provide a list of the publications derived from this work:

1. Ahmet Firintepe, Mohamed Selim, Alain Pagani, and Didier Stricker. The More, the Merrier? A Study on In-Car IR-based Head Pose Estimation. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1060–1065, IEEE, 2020.
2. Ahmet Firintepe, Alain Pagani, and Didier Stricker. HMDPose: A large-scale trinocular IR Augmented Reality Glasses Pose Dataset. In *26th ACM Symposium on Virtual Reality Software and Technology (VRST)*, ACM, 2020.
3. Ahmet Firintepe, Alain Pagani, and Didier Stricker. A Comparison of Single and Multi-View IR image-based AR Glasses Pose Estimation Approaches. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 571–572, IEEE, 2021.
4. Ahmet Firintepe, Carolin Vey, Stylianos Asteriadis, Alain Pagani, and Didier Stricker. From IR Images to Point Clouds to Pose: Point Cloud-Based AR Glasses Pose Estimation. *MDPI Journal of Imaging*, vol. 7, no. 5, 2021.
5. Ahmet Firintepe, Oussema Dhaouadi, Alain Pagani, and Didier Stricker. Comparing Head and AR Glasses Pose Estimation. In *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2021.
6. Ahmet Firintepe, Sarfaraz Habib, Alain Pagani, and Didier Stricker. Pose Tracking vs. Pose Estimation of AR Glasses with Convolutional, Recurrent, and Non-Local Neural Networks: A Comparison. In *EuroXR 2021: Virtual Reality and Augmented Reality*, Springer International Publishing, 2021.
7. Mohamed Selim, Ahmet Firintepe, Alain Pagani, and Didier Stricker. AutoPOSE: Large-Scale Automotive Driver Head Pose and Gaze Dataset with Deep Head Pose Baseline. In *International Conference on Computer Vision Theory and Applications (VISAPP)*, SCITEPRESS Digital Library, 2020.

2 Background

It is imperative to convey a fundamental understanding of the foundations of the thesis topic to comprehend its aim and contributions. Since the work is rooted in the areas of Augmented Reality, Deep Learning and pose estimation, we give a more detailed introduction and overview on the subjects.

In the following, the term ”Augmented Reality” is defined, explained, and differentiated first. Then, its potential use in the car context will be further presented.

In the second section, Machine Learning and Deep Learning is introduced. The most commonly types of Deep Neural Networks like Convolutional Neural Networks and Recurrent Neural Networks are presented. A short overview of Deep Learning networks used in Computer Vision concludes this section.

Finally, various possible pose estimation and tracking types will be explained, ranging from Deep Learning and feature-based pose estimation to head and object pose estimation to inside-out and outside-in pose estimation. Concluding with pose estimation techniques applied in the automotive context, the need for 6-DoF pose estimation of AR glasses and the head will be presented.

2.1 Augmented Reality

With the rise of Virtual and Augmented Reality, different terms of this context are being mentioned more often in the media. Sometimes, terms like ”Mixed Reality”, ”Augmented Reality”, or even ”Virtual Reality” are wrongly used interchangeably. Thus, the term AR will be defined and differentiated from other areas by using the reality-virtuality continuum. In this continuum, the previously mentioned related areas will be explained. Then, the components of a system that enable AR will be described. We finally present AR systems that are deployed inside cars.

2.1.1 Definition and Differentiation

The most common definition of Augmented Reality was introduced by Azuma [15]. He defined AR as a system with three properties:

- Combination of reality and virtuality.
- Interactive in real-time.
- Registration in a three-dimensional space.

Since AR is commercially rediscovered recently, various new, less technical, and more straightforward definitions have been formulated. Schart et al. [16] describe Augmented Reality as the perception of reality augmented with virtual elements.

To differentiate the term Augmented Reality clearly from related fields, the reality-virtuality continuum has been introduced, and other terms in this setting have been defined.

Reality-Virtuality Continuum

Milgram et al. [17, 18] introduced the reality-virtuality continuum to distinguish between all manifestations between virtuality and reality (Figure 2.1).

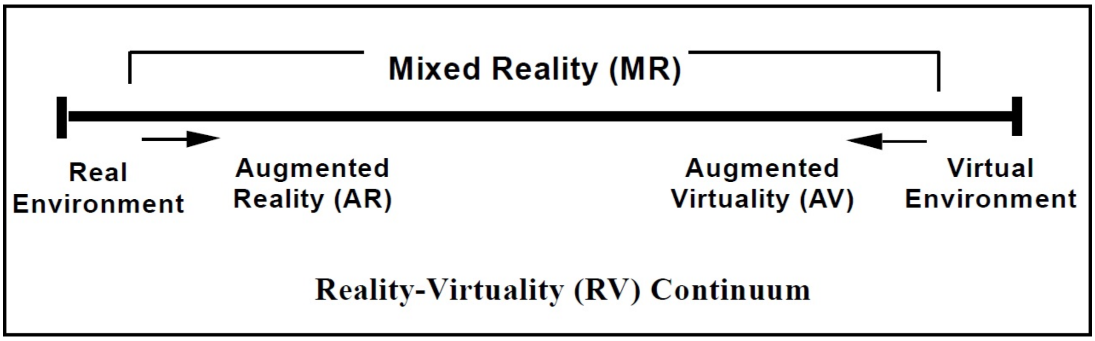


Figure 2.1: The Reality-Virtuality Continuum [18].

At one end of the continuum, there is the reality without any virtual content. At the opposite end of the continuum, virtuality with exclusively virtual elements exists, also known as ”Virtual Reality” (VR).

Virtual Reality specifies itself as the usage of computer technology to create the effect of an interactive and three-dimensional world, in which the objects possess a sense of spatial presence [19]. Bryson [19] describes ”sense of spatial presence” as the perception of objects by the viewer in the virtual space independently from the user and the display technology with a given spatial position. All manifestations, which emerge from the combination of reality and virtuality, are called Mixed Reality. The umbrella term also includes the previously defined term Augmented Reality and ”Augmented Virtuality” (AV). Augmented Reality exists, if the reality is being extended with virtual elements with properties like interactivity in real-time and registration in three-dimensional space. In return, Augmented Virtuality applies if virtuality is being augmented with real objects. Metzger defines AV as the largely virtual environment, which is extended with real objects [20].

Another term related to AR, sometimes seen as a subterm of AR, is ”Diminished Reality” (DR). Diminished Reality defines itself as using AR technology to reduce the number of perceived elements [21]. Wassom justifies the subordination of the term to AR, as the filtering of elements of the real world augments the real environment in a qualitative manner [21].

AR and DR are both based on the same technical components, independent of their specific software processing.

Components of Augmented Reality Systems

Three elements in a system enable Augmented Reality [22]. The used system must enable tracking first. Tracking is defined as capturing the surroundings to extend the real world with virtual objects [23]. In case tracking has been enabled, virtual objects can be placed in the environment. To manipulate the placed objects, interaction with the system and, thus, with the virtual objects must be ensured. To finally perceive the virtual content, they have to be shown on a display.

AR Displays

Two ways of categorization for displays exist that enable Augmented Reality. On the one hand, it is possible to subdivide them by their form of display. On the other, they are categorized regarding their usage.

Display Categories by their Form of Display Displays can be divided into two categories by their way of displaying the content [22]. On the one hand, there are video see-through displays, where the camera image of the surrounding with the extension of virtual content is shown [24]. This display form is typically included in smartphones or tablets and enables a high quality of augmentation of the real world with virtual elements.

On the other hand, there are optical see-through displays. These displays are semi-transparent to see the real world behind the display [25]. By showing virtual objects on the display, the virtual content can be perceived ideally as part of the real world. Data glasses usually consist of optical see-through displays.

Display Categories based on their Usage Another possibility to divide displays into categories is by their way of usage. The categories are handheld displays, head-mounted displays, head-up displays, and contact lenses [23].

Handheld Displays: Handheld displays typically consist of a video see-through display and a computing unit while being held in the user's hand [25]. A popular example of this category is smartphones.

Contact Lenses: It is possible to integrate displays into contact lenses. In this case, the graphical content must be transmitted from another device, as lenses cannot contain any computing unit [26].

Head-Up Displays: This type of display is placed in the user's surroundings, where an image is projected on a semi-transparent surface [16]. Subsequently, they are optical see-through displays, often used in the automotive context.

Head Mounted Displays: Head-mounted displays are displays that are being attached to the head [24]. They are often in the shape of a helmet or data glasses. They can be based on either optical see-through displays or video see-through displays.

In the car context, the most popular way of enabling AR is head-up displays. In recent years, the deployment of smart glasses has become an exciting option for the car.

2.1.2 In-Car AR Systems

In recent years, various ways of in-car AR systems have been either conceptualized or have already been deployed in real cars. They can typically be divided into head-up displays, AR head-up displays, and AR glasses.

Head-Up Display

The oldest and simplest form of an AR system is head-up displays (Figure 2.2).

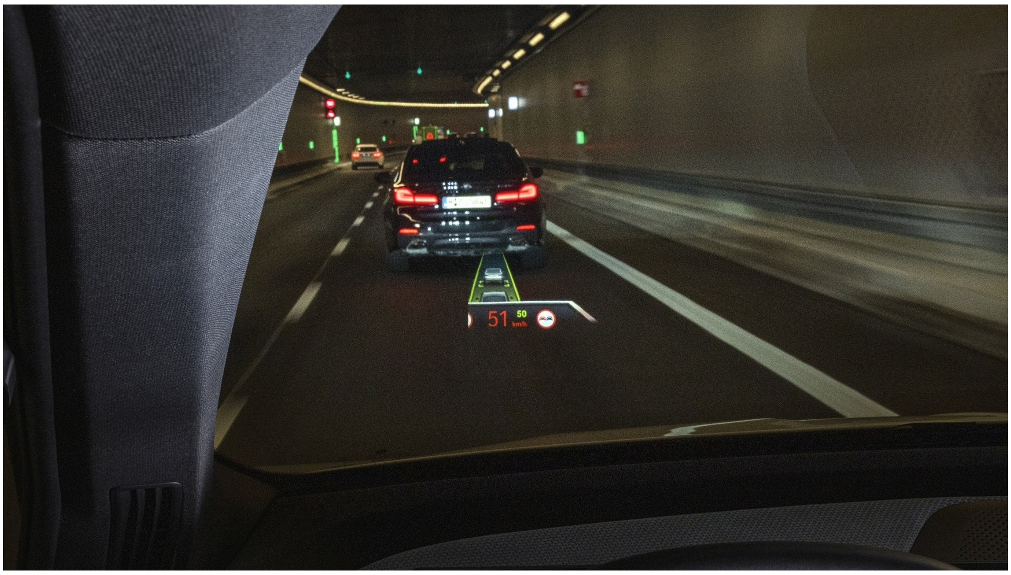


Figure 2.2: Head up display of the BMW iX [27].

They offer to show virtual content directly in the view of the driver. The display area is usually restricted to a small area. Typical use cases shown in head-up displays are warnings, speed, speed limit, or intractable content like a switchable music playlist. This type of head-up display usually does not contain 3D content that is registered in the real environment. Subsequently, the content is shown static to the car movement. It is generally applied to show 2D information. Thus, the displayed information does not fully meet all requirements of the AR definition.

AR Head Up Display

More recently, Augmented Reality head-up displays have been introduced into end-consumer cars (Figure 2.3). In this category, the displaying area is more extensive, offering more information. The content can be registered in 3D in the real world, enabling use cases independent from the car movement. Usually, as the displaying area is still restricted, the information is partially

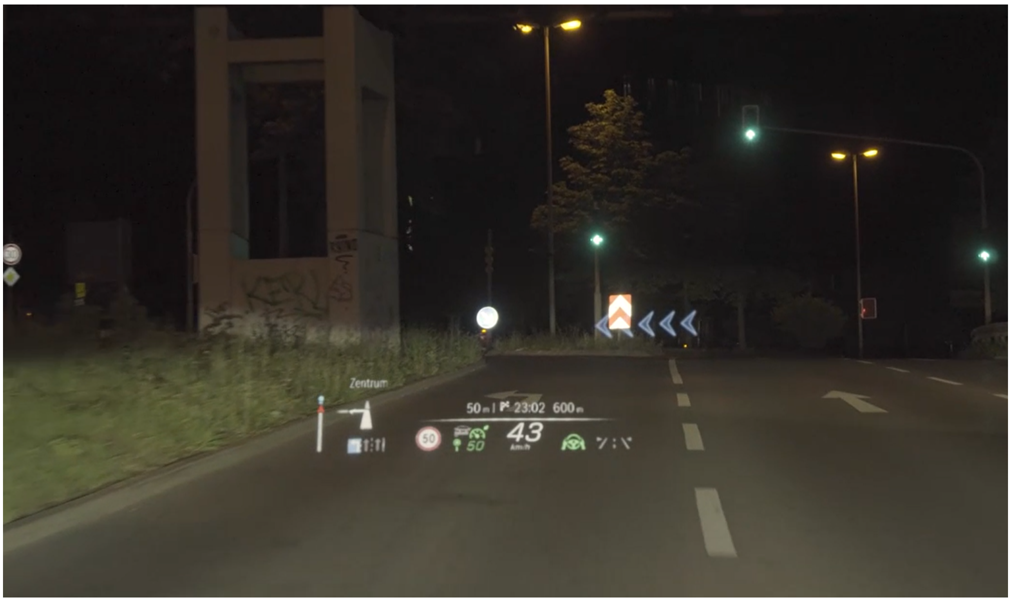
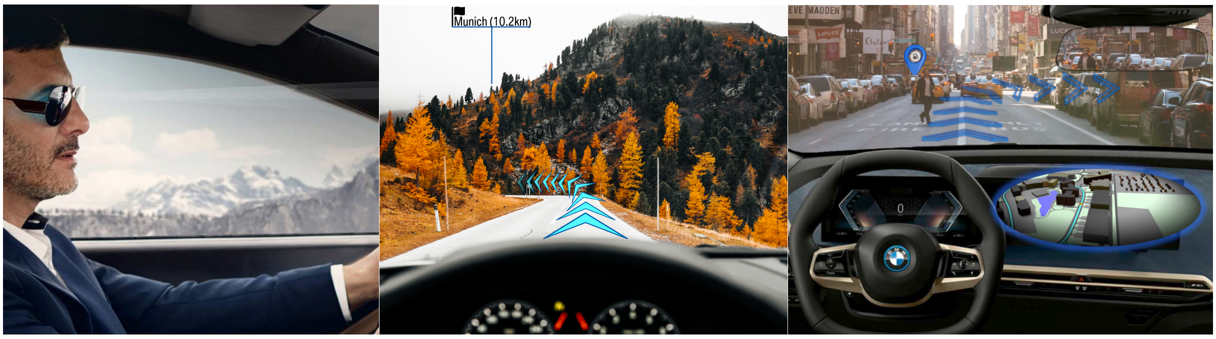


Figure 2.3: AR head up display of the Mercedes S-Class [28].

static to the car movement and intermixed with 3D, world registered AR navigation arrows. This type of head-up displays fulfill the AR definition, but due to their restricted display area, which is fixed to a certain area in the car, the amount of 3D use cases that can be deployed is limited.

AR Glasses

Another, more recent development regarding AR inside cars is the usage of smart glasses (Figure 2.4).



(a) AR glasses used inside the car. (b) AR navigation and points of interest highlighting. (c) Potential use cases inside and outside the car.

Figure 2.4: Visualizations for in car AR glasses usage.

AR glasses enable a wide variety of use cases inside and outside the car while driving. The main advantage of a head-mounted display over a head-up display is the existence right in front of the wearers' eyes regardless of their head movement. By this, typical outside use cases like AR navigation and point of interest highlighting are possible. In addition, inside use cases can be developed, like displaying a 3D map of the surrounding area or a 3D avatar as an assistant.

The advantage of a free-moving display with the head by using AR glasses comes with the necessity to accurately and precisely measure the AR glasses or the head pose at all times. This is conventionally conducted with a fusion of IMU measurements built inside the glasses and inside-out tracking based on built-in cameras. This approach is not compatible with the in-car scenario, which will be further discussed in Subsection 2.3.4.

2.2 Machine Learning and Deep Learning

Measuring the position and orientation of objects in recent years is typically conducted based on Machine Learning or Deep Learning algorithms. This section introduces Machine Learning and Deep Learning first. After presenting common types of Deep Learning, typical Computer Vision use cases and popular algorithms round off the section.

2.2.1 Machine Learning

In 1959, Arthur Samuel [29] coined the term Machine Learning. Tom Mitchel defined the term Machine Learning as follows: "Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience" [30]. In contrast to Machine Learning, traditional algorithms are hand-tailored by developers to solve problems, whereas in Machine Learning, the computer is taught to solve a specific problem based on data.

There are different types of Machine Learning algorithms, which can be clustered into three categories called Supervised Learning, Unsupervised Learning, and Reinforcement Learning:

Supervised Learning: In Supervised Learning, a labeled dataset is given, with which the algorithm can be trained to solve a problem. The model can measure the accuracy between its prediction and the ground truth by using labeled data, thus learning to solve the problem accurately over time. The two standard tasks of Supervised Learning are classification and regression:

Classification: In a classification task, the algorithms learn to distinguish between discrete categories. One typical example is the determination of whether a cat or a dog is visible on an image.

Regression: A regression task requires the model to output a continuous value. One characteristic example is housing prices, given various parameters like the type of housing, size, and the number of rooms. Another example is pose estimation, which is the focus of this work.

Unsupervised Learning: Algorithms that learn on data without ground truth labels are part of Unsupervised Learning. One typical task is clustering, where the data is learned to be separated into clusters. Unsupervised Learning is mainly applied to identify structures and discover patterns in data to obtain valuable insights into the data at hand without having any labels available.

Reinforcement Learning: In these algorithms, an agent takes actions to reach predefined goals by interacting with an environment. The agent receives a reward based on its action based on how good the action was to achieve the goal. This could be driving a vehicle in a simulation or learning to play a game.

Machine Learning encapsulates classical and Deep Learning algorithms. Classical algorithms usually are more dependent on human intervention to learn. Typically, the amount of data to learn from is limited, requiring more structured data to learn. Another subset of Machine Learning is Deep Learning. The term Deep Learning was first introduced by Rina Dechter in 1986 [31]. Deep Learning models are able to work with more unstructured data to solve more complex problems, requiring a vast amount of data at the same time. Deep Learning algorithms usually deploy Deep Artificial Neural Networks.

2.2.2 Deep Artificial Neural Networks

Deep Learning methods comprise artificial neural networks with multiple layers to perform mentioned tasks. Artificial neural networks are inspired by the idea of neurons in a brain. Subsequently, the basic building block of a neural network is an artificial neuron. One neuron can be interpreted as a processing unit. One hidden layer in a network typically contains multiple neurons. If - in addition to an input layer representing the input data and the output layer outputting the estimation of the model that learned a specific task - multiple hidden layers exist in a neural network, they are called Deep Neural Networks (DNNs). This type of network, where all neurons of one hidden layer are connected with all neurons of the next hidden layer, is called multilayer perceptron or fully connected network. Figure 2.5 shows a standard DNN architecture.

DNNs are currently the most popular Machine Learning-based algorithms to solve problems. They are trained by the usage of a loss function, which is computed in the output layer. The loss function compares the network's output with the ground truth label and updates the rest of the network accordingly, representing the network's learning process. An optimization algorithm defines how the network parameters are updated after each learning step. Two popular DNNs used in Computer Vision and Natural Language Processing are Convolutional Neural Networks and Recurrent Neural Networks.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is the most popular type of Deep Neural Network. It is often utilized to process visual data. They are based on the previously mentioned multilayer

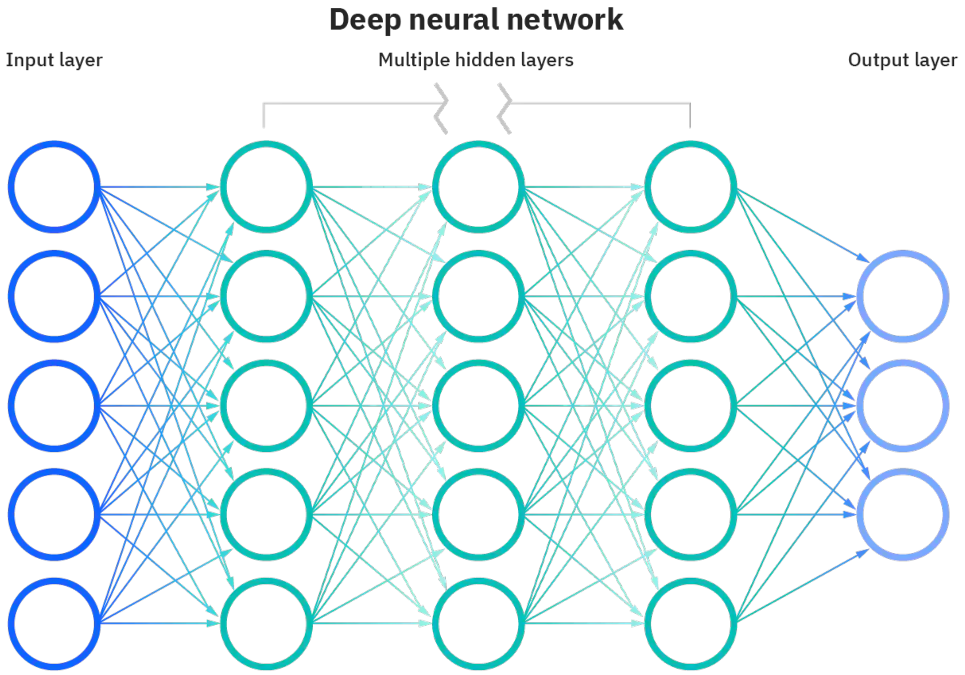


Figure 2.5: A standard DNN with an input and output layer, with multiple hidden layer between them [32].

perceptron architecture. Instead of using fully connected hidden layers, CNNs utilize mainly convolutional layers. In contrast to connecting all neurons of one layer with all neurons of the next layer, a neuron of a convolutional layer is only connected to a limited amount of close neurons, where the same set of weights is used. This reduces the number of parameters compared to fully connected layers. CNNs use these shared kernel weights to map low-level features in the previous layer to high-level features in later layers.

CNNs are a popular choice in Computer Vision. In traditional Computer Vision algorithms, essential features in images were hand-crafted and then learned. CNNs automate this learning and feature extraction process without human participation. In order to have a data reduction from previous layers to later layers, an operation called pooling is often performed. Typically, fully connected layers are deployed towards the end of the network, mapping the extracted, high-level network features to concrete values of the task at hand.

Figure 2.6 shows an example architecture of a CNN with a classification task. The training of a CNN is performed as already described for DNNs in the beginning of this Subsection 2.2.2.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) differentiate from other neural networks by modeling the temporal characteristics of sequential data. RNNs have internal memory units to store former

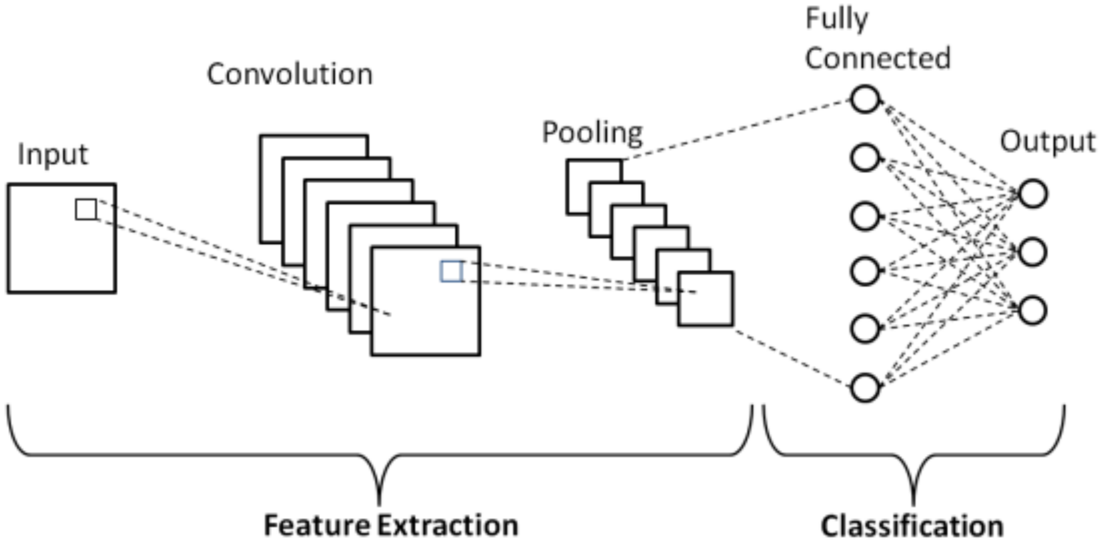


Figure 2.6: CNN architecture with a classification output [33].

samples of data in order to learn sequential patterns. They aim to identify trends in data sequences, which typically occur in audio, text, and video.

Unlike feed-forward neural networks like pure CNNs, RNNs take the time into account to utilize previous information. An RNN cell incorporates a hidden state, representing the information from all the previous steps. Depending on the current input and the information stored from previous states, a prediction can be performed. Figure 2.7 shows a comparison between a feed-forward neural network and an RNN.

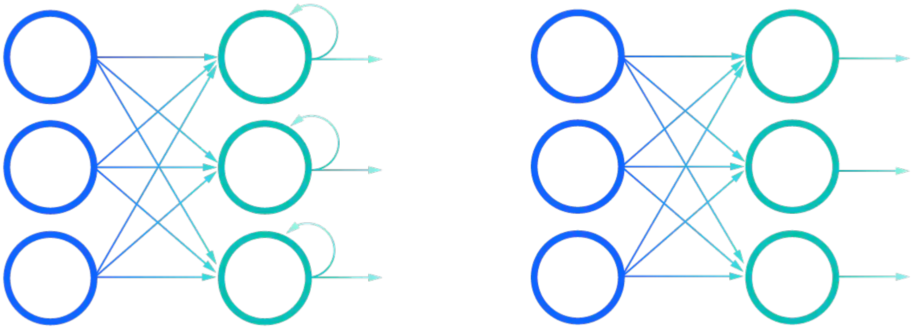


Figure 2.7: Comparison of an RNN (left) and a feed-forward neural network (right) [34].

Since all timesteps share the parameters of RNN cells in the network, the gradient in each output depends on a potentially long series of timesteps. This comes with a problem for the cells: if the weights are too small, it can lead to vanishing gradients where gradient values per layer get small, interrupting training. This problem is amplified by RNN cells having difficulties with long-term dependencies. To counter these problems, Long Short-Term Memory units (LSTMs) were proposed [7]. LSTMs retain information over more extended time series and are an extension of basic RNN cells. Basic RNNs do only have hidden states, which serve as the memory. LSTMs, on the contrary, have both cell states and hidden states. The cell state can remove or add information to the cell, enabling long-term dependencies.

RNNs and CNNs are utilized in a variety of Computer Vision tasks, where they achieve outstanding performances. They are presented in the following.

2.2.3 Deep Learning in Computer Vision

Deep Learning gained popularity for Computer Vision tasks due to its successful application. According to Huang et al., "...computer vision aims to build autonomous systems which could perform some of the tasks which the human visual system can perform..." [35]. Consequently, Computer Vision is being deployed in numerous fields, ranging from Social Media over Robotics to Autonomous Driving. More specifically, some characteristic Computer Vision tasks are the following:

Image Classification: Image classification describes the task of classifying an image. Typical examples are a dog or cat classifier or a person detector. In general, image classifiers are able to predict a particular class of an image accurately.

Object Localization: This category is similar to image classification. Instead of classifying images, the main objective is to derive the location of an object in an image without classifying it. Object pose estimation can be considered in this category if the 6-DoF-pose of an object is regressed per image.

Object Detection: Object detection combines the previous two categories: an object is classified and its location is predicted.

Object Tracking: In object tracking, an object is followed or tracked over sequences. It is typically deployed regarding vision with videos. Object pose estimation can also be included in this category, as the 6-DoF pose of an object is tracked over a period of time. If one or more of the previous images are taken into account, the task is certainly called object tracking.

Recently, mostly CNNs and RNNs have been deployed for Computer Vision tasks. Over the years, several Deep Neural Network architectures have been developed for vision tasks.

AlexNet AlexNet [36] is one of the first CNNs used in Computer Vision. It was first developed for image classification in 2012. Its success influenced the Computer Vision community to use CNNs more intensely for vision tasks. Figure 2.8 shows an overview of its architecture. In comparison to recent architectures, it has a limited amount of five convolutional layers, and three consecutive fully connected layers.

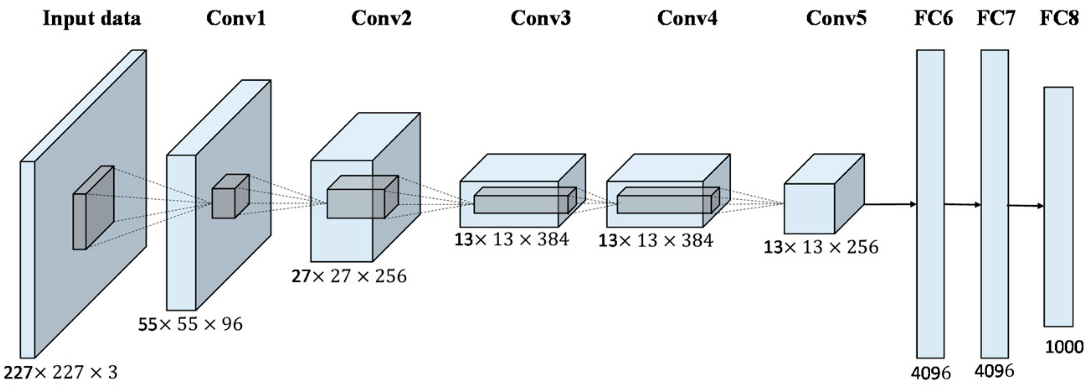


Figure 2.8: Overview of the AlexNet architecture [37].

VGG Simonyan et al. [38] introduced the VGG network in 2014, which depicts a typical CNN with consecutive fully connected layers. Figure 2.9 presents the 16 layer variant.

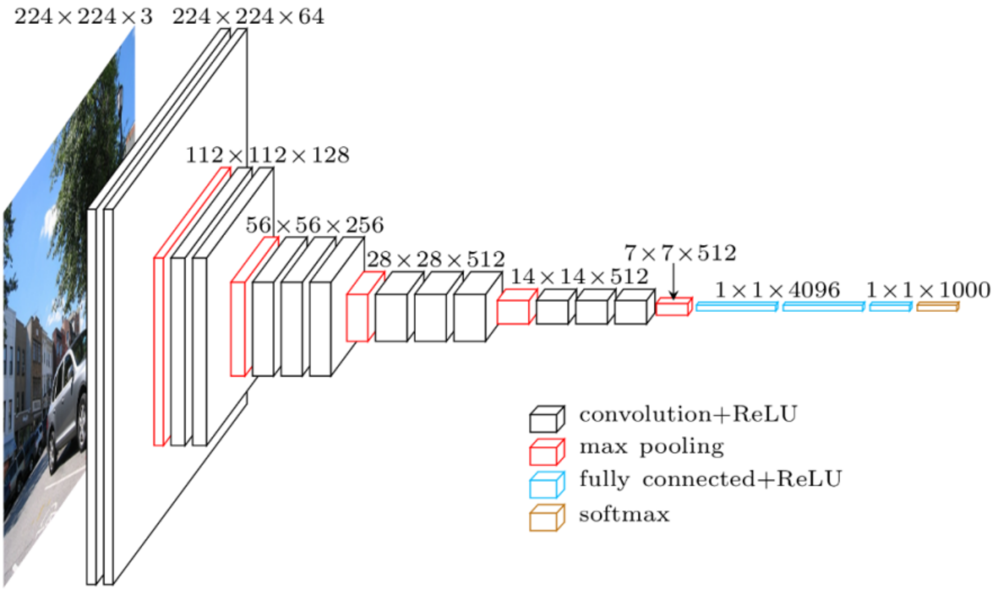


Figure 2.9: Overview of VGG16, which is the 16 layer variant of VGG [39].

The figure shows that the network includes 13 convolutional layers and three fully connected layers, with max pooling layer to reduce the dimensionality.

Inception The Inception network [40] was developed by Google in 2014 as an alternative to handpick the filter sizes of convolutional layers. Instead, they introduce inception modules, in which an input is first being downscaled to save computation. Then, convolutions with varying filter sizes are being applied and concatenated. This output forms the input for the next inception module. Figure 2.10 exhibits an overview of the architecture, highlighting an inception module. This is done over multiple blocks until a value is being estimated.

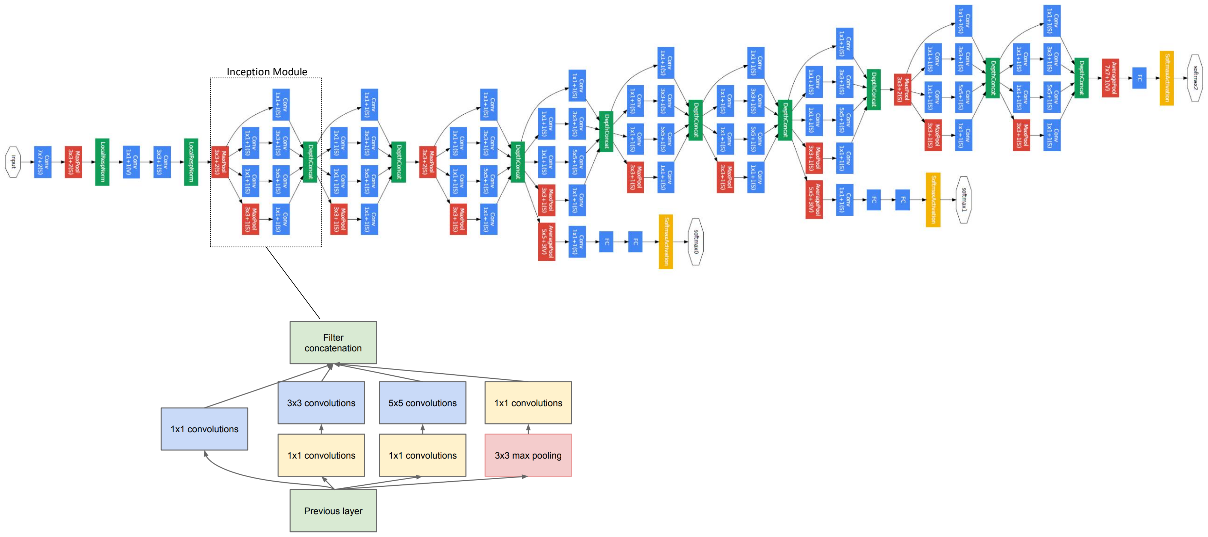


Figure 2.10: Overview of the Inception architecture [40].

ResNet Hu et al. introduced ResNet [3] in 2016, paving the way for a new generation of Deep Learning algorithms. ResNet can include hundreds of layers, while the layers are grouped into blocks of 2 or three layers, depending on the architecture size. Some of them are residual blocks, which introduce residual connections from the input of a block to the output of the same block. Figure 2.11 illustrates the ResNet architecture by comparing its counterpart without residual connections and to VGG. Typically, deeper CNNs usually mean higher estimation performance, but in reality, they come with a degradation problem. The degradation problem means that stacking more layers can degrade the estimation performance as they converge at higher error rates. ResNet solves this problem by adding the mentioned residual blocks. Furthermore, they help reduce the vanishing gradient problem, as the information is kept for longer thanks to the residual connections.

All presented networks were initially developed to classify images. All of them were later adapted and utilized as a backbone in several tasks including regression. One of the most interesting regression tasks is object pose estimation and tracking.

2.3 Pose Estimation and Tracking

6-DoF pose estimation and tracking are often used interchangeably but generally describe two different ways of regressing the pose of an object. 6-DoF pose estimation usually contains frame-by-frame pose estimation methods that take input in the form of an image or point cloud

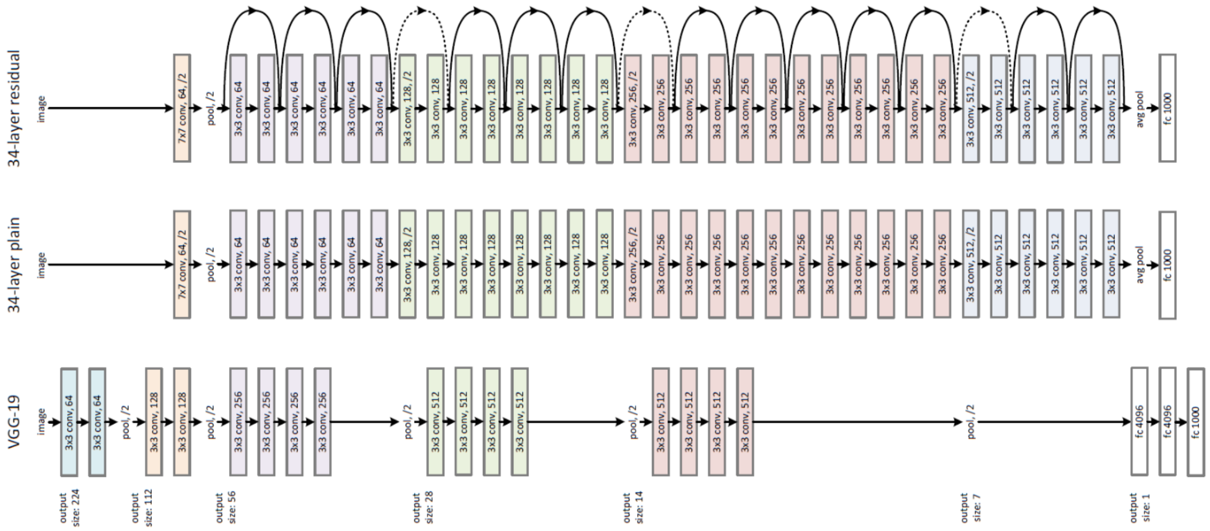


Figure 2.11: Overview of the ResNet-34 architecture on top. On the bottom, VGG-19 is displayed for comparison. In the center, the ResNet model without residual blocks is depicted [3].

at one timestep and subsequently estimate the pose. In contrast, pose tracking represents algorithms that take previous timesteps into account to estimate the current pose.

In this thesis, pose tracking methods are specifically emphasized whenever they are used. Most approaches presented in this thesis are frame-by-frame pose estimation techniques. Both ways of regressing the pose can be either based on feature-based, non-learning techniques or Deep Learning-based methods.

2.3.1 Deep Learning vs. Feature-based Pose Estimation

Traditional Computer Vision approaches for image-based object pose estimation mainly focused on extracting features from the input image, matching them with the features of the object model, and finally using the 2D-3D point correspondences to perform Perspective-n-Point (PnP) to obtain a pose [41, 42]. In recent years, Deep Learning-based methods showed significant improvement for pose estimation, where two main categories exist. In the first category, methods directly regress the pose based on the input image [43, 44, 45]. Xiang et al. proposed PoseCNN, which predicts a segmentation mask of the object, the depth, and the rotation, which they then combine to estimate the 6-DoF pose [43]. Other approaches, which partly rely on the traditional branch of using PnP, use deep neural networks to estimate the keypoints of the objects. Then, they perform PnP on the given 2D-3D keypoint correspondences to regress the pose [46, 47, 48]. Peng et al. [48] presented a hybrid approach called PVNet between the two mentioned categories, which is also based on keypoint regression and further PnP execution. A novelty in their approach is the regression of 2D vectors from each pixel to predefined keypoint positions of the object with a CNN. The ground truth 3D keypoints for the PVNet method are generated using 3D models of the target objects. Based on this vectorfield output, they conduct a voting-based approach to estimate to most probable keypoint position and its uncertainty as a covariance matrix. Then, they perform PnP [48].

2.3.2 Head Pose Estimation

Head pose estimation is a field that enjoyed great popularity in the Computer Vision community. In this field, traditional and Deep Learning-based methods have been conducted, while the latter dominated in recent years. Layher et al. [49] present a pipeline for traditional head pose estimation. Based on stereo grayscale images, they extract the bounding box of the head and facial features jointly using a Support Vector Machine (SVM). Then, the features are being stereo-matched with a subsequent 3D reconstruction with the cameras' extrinsic and intrinsic parameters. Finally, a plane defines the head pose, fitting four facial 3D landmarks.

Deep Neural Networks have recently proven useful in various head pose algorithms [50, 51, 52, 53, 54]. The methods mainly use CNNs to utilize their generality performance and obtain an accurate way of estimating the pose. Cao et al. [53] use RGB images to feed them into a ResNet-50 [3] backbone. The use features extracted at different layers in the ResNet backbone to regress the head's orientation further. In contrast to the state-of-the-art that represents the orientation mostly in either Euler angles or quaternions, they opt for rotation matrices to avoid discontinuity and ambiguity, despite its redundant properties. After the backbone extracts the features, they are utilized by the prediction module, which includes a capsule network [4] to regress an orientation.

Instead of using an RGB image, Gupta et al. [55] introduced a high-level features representation in their model as input for the proposed network. In the input feature maps, five facial keypoints heatmaps are included representing the probability distribution of the keypoint locations. By this, they achieve a significant prediction error reduction in comparison to other head pose estimation methods.

2.3.3 Object Pose Estimation

As introduced in Subsection 2.3.1, 6-DoF pose estimation approaches can primarily be divided into two categories. In the first category, 2D targets and 2D-3D object correspondence are detected first to further solve the Perspective-n-Point (PnP) problem to obtain the 6-DoF pose. Approaches in the second category conduct direct pose regression by inputting the image directly into a Deep Neural Network to estimate a pose.

The first category consists of approaches that first detect the targeted object in an image. Then, they solve a PnP problem for the 6-DoF pose. Keypoint-based methods and 2D-3D correspondence-based methods exist in this category. Methods based on keypoints either directly predict the 2D projections from 3D models of an object using the furthest point algorithm [48, 56, 57], or use the keypoints from the object's surface [47, 46, 58]. The dense 2D-3D correspondence methods estimate the matching point 3D point on the model for each object's pixel. They use them for PnP to get the 6-DoF pose [59, 60, 61].

Various 6-DoF pose estimation techniques have been introduced to regress the pose of an object based on direct regression on the image information only [62, 43, 63]. Kendall et al. introduced PoseNet [62], a deep CNN to estimate for 6-DoF camera pose estimation based on a single RGB image. The network regresses the pose end-to-end in real-time. They motivate the network by referencing experiments, where they trained individual networks to estimate the position and orientation separately. That method performed worse compared to their training with full 6-DoF pose labels.

Another popular example is PoseCNN [43], which decouples the problem of pose estimation into separately estimating the translation and orientation. It is based on a pretrained VGG-backbone [38], which they use for feature extraction and further implement three output branches. Two fully connected branches estimate the center direction and the depth of every image pixel, respectively. The third branch comprises ROI pooling and a fully connected architecture. It estimates a quaternion for the rotation per region of interest.

Capellen et al. introduced ConvPoseCNN [63], which further extends PoseCNN [43]. It rather performs pixel-wise quaternion prediction instead of ROI-based orientation prediction. They conduct no further changes to the translation and ROI of PoseCNN [43].

2.3.4 Inside-Out and Outside-In Pose Estimation

There are two types of visual tracking methods, which are both used in the VR and AR context. One is Inside-Out tracking, and the other is Outside-In tracking [64, 65] (Figure 2.12).

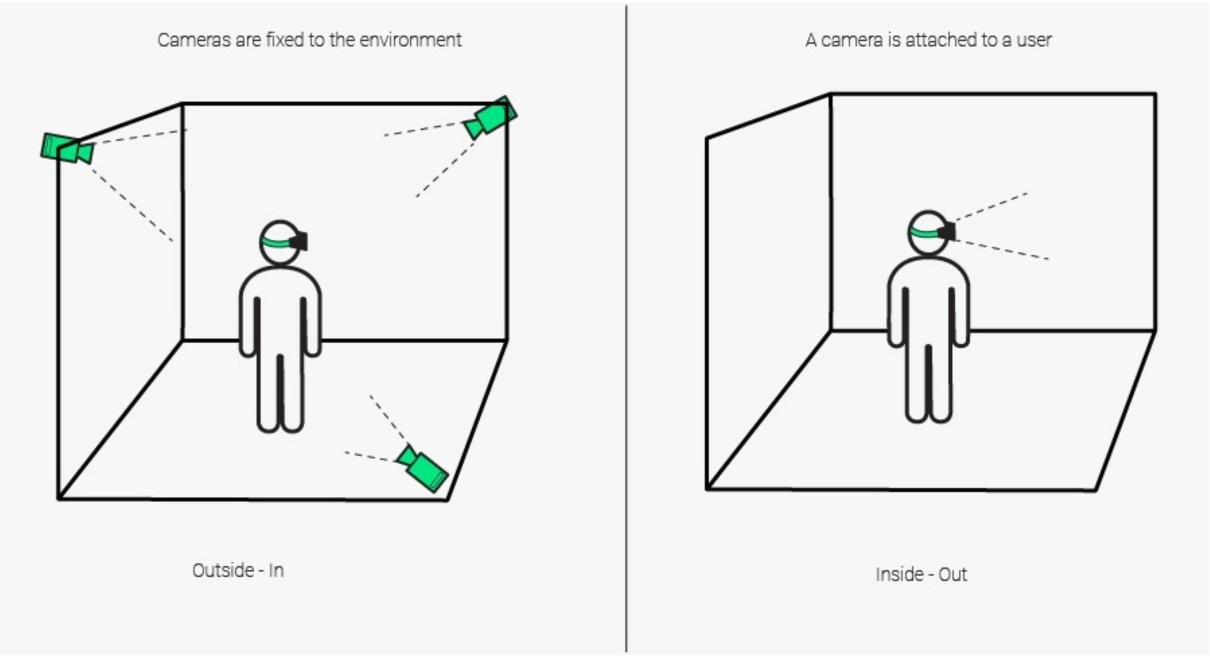

Cameras are fixed to the environment
A camera is attached to a user
Outside - In
Inside - Out

Figure 2.12: Inside-Out vs. Outside-In Tracking [66].

Inside-Out Tracking Inside-Out tracking describes the performing of tracking through one or more cameras build inside the moving object. In the AR glasses context, most tracking enabled glasses perform Inside-Out tracking as in most cases there are cameras build inside the glasses. They can be utilized to locate the glasses in their environment. As Inside-Out tracking requires sensors built inside the HMD and enough computing power on it to deploy an adequate tracking algorithm, an alternative for light-weight glasses can be Outside-In tracking.

Outside-In Tracking Outside-In tracking is a method of mounting multiple sensors in the environment of the observed object. This is not efficient for an open world, outdoor scenario as it requires a considerable amount of sensors, but it can make sense for indoor purposes. Especially scenarios where AR glasses are used inside a car can benefit from this as there is limited space. Furthermore, Inside-Out tracking in an In-Car scenario is challenging as a driver mostly sees dynamic information from the outside world and static information from the car interior. This makes it more difficult to deploy algorithms that require to be robust against the presence of static and dynamic information simultaneously.

2.3.5 Pose Estimation Algorithms in the Automotive Context

In the driver assistance context, there have been many recent works regarding pose estimation. In the autonomous driving context, it is crucial to regress the vehicle's pose on the street, especially in urban areas [67, 68, 69].

Another automotive case for pose estimation is the 6-DoF head pose of the driver [70, 71, 72, 73]. Borghi et al. proposed POSEidon [71], an approach based on depth images for head pose estimation. They perform head cropping using bounding box positions, which are first being estimated. In the same architecture, they place a network to generate gray-level images from depth faces. By deploying the Farneback algorithm [74], a motion image is being derived. They use the motion image, the cropped image, and the produced gray-scaled image to fuse and feed them into a Fully Connected Network to regress the head orientation. This approach performs robustly in extreme rotations, in the presence of occlusions, and under varied lighting conditions. Borghi et al. [72] propose another method that introduces Recurrent Neural Networks in addition to depth information.

Hu et al. [73] designed a Deep Learning network based in point cloud data for head pose predictions, using a depth camera to construct point cloud data. Points of the cloud are sampled and grouped by a sequence of abstraction layers, where PointNet++ [6] is utilized to capture local features in each layer. In the end, they use a Fully Connected Network to regress the pose based on the extracted discriminative features.

Finally, pose estimation is a key component to enable HMDs while driving. For this purpose, an exact pose of the worn AR glasses is needed. The pose is used to project the virtual AR content at the right position in the world, regardless of the constant car movement and the movement of the wearers' head. An exact head pose could help stabilize this projection on top of the HMD pose. There are multiple error sources in an AR system beyond the pose estimation error contributing to the overall error, affecting the accurate display of the virtual elements.

3 Error Ranges for Augmented Reality Car Use Cases

AR Systems generally consist of various software and hardware components. The installation and the calibration of the hardware components introduce errors to the overall system. In an AR system, multiple error sources from the placement of the tracker, to tracking estimation accuracy, to rendering the final image exist. All errors affect the credible and realistic superimposition of the real world with virtual content.

This chapter first defines and explains potential errors in our AR system based on general error definitions for AR systems [75, 76]. Subsequently, we present some errors which are invariant in our specific car setup.

Then, we define important error ranges based on car use cases for the 6-DoF pose estimation of AR glasses or the head of the driver, respectively. They are categorized regarding their superimposition distance in AR to the viewer.

3.1 Errors in Augmented Reality Systems

One of the most critical problems in AR systems is when the virtual objects are not correctly registered with the real world environment and may appear misaligned [75]. This discrepancy is called registration error and is measured by computing the distance between the position the virtual object is meant to be positioned at and its real position in 3D world space. The registration error is often used to obtain the quality of an AR system.

Holloway [75, 76] defines various errors in Augmented Reality Systems, which all play into the registration error. They focus on a setup for see-through Head-Mounted-Displays, which is directly applicable to our existing setup, where we deploy smart glasses. In general, the errors can be split into Acquisition/alignment error, Head-tracking error, Display error, and Viewing error [76].

3.1.1 Acquisition Error

If an object of the real world is first acquired to be later shown on the display, a 3D-scan of the target object is required. This procedure introduces errors, as the 3D-scan of the target objects is prone to errors. The introduced error through the digital acquisition of the real target object is called acquisition error. This error is not being targeted in this thesis as we assume virtually generated 3D content for later display.

3.1.2 Head-Tracking Error

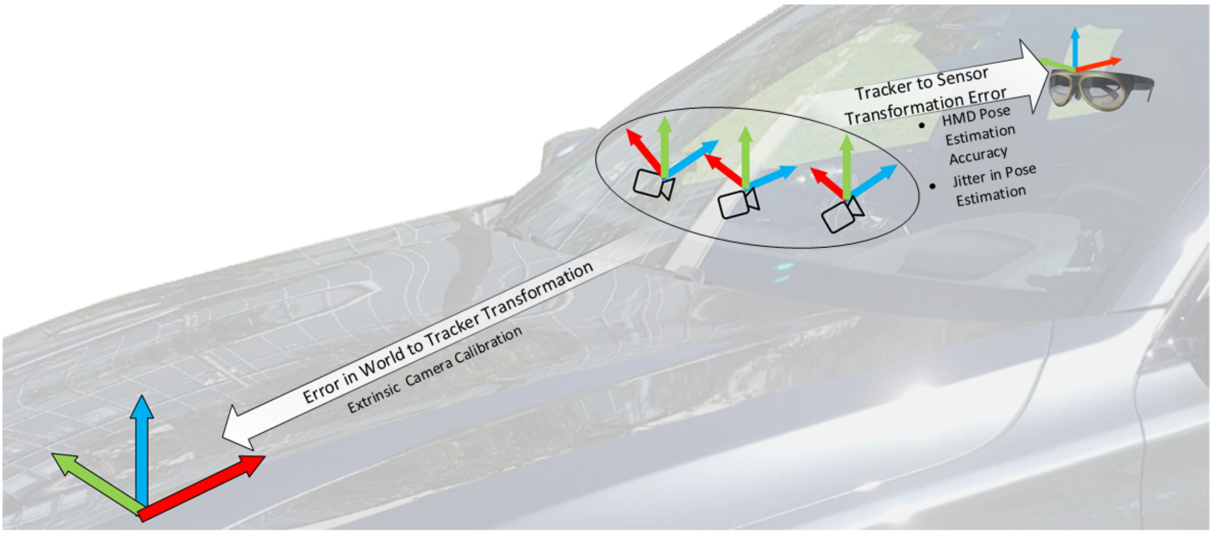

Tracker to Sensor
Transformation Error
HMD Pose Estimation Accuracy
Jitter in Pose Estimation
Error in World to Tracker Transformation
Extrinsic Camera Calibration

Figure 3.1: Visualization of the two main head-tracking error sources with concrete examples in our AR system.

The head-tracking error consists of multiple error sources and is the major cause of the registration error in AR systems [75]. It describes the error introduced by tracking the position and orientation of the head or HMD in the world. More specifically, Holloway [75, 76] defines it as the error in the World-Tracker and Tracker-Sensor transformations. Figure 3.1 shows both errors in our setup with concrete examples.

Error in World to Tracker Transformation

In case of the initial definition by Holloway, the setup assumes a tracker on the Head-Mounted-Display to conduct inside-out tracking. Thus, the world to tracker transformation describes the tracker position and orientation of the tracker in world coordinates, which is measured once. In our case of outside-in pose estimation, this transformation is similar: it fundamentally describes the error introduced by calibrating of the tracking cameras to the world coordinate system. The error is defined as follows:

$$e_{W\_T} = ||\delta \mathbf{v}_{W\_T}|| + 2 \cdot \sin \frac{|\delta \varnothing_{W\_T}|}{2} \cdot (||\mathbf{v}_{T\_S}|| + ||\mathbf{v}_{S\_P}||) \tag{3.1}$$

Here, $\delta \mathbf{v}_{W\_T}$ describes the positioning error the tracker's origin T in world coordinates W. $\delta \varnothing_{W\_T}$ is the orientation error of the tracker T in the world W, which are the cameras in our case. The vector $\mathbf{v}_{T\_S}$ from the origin of the tracker T to the origin of the sensor S, and $\mathbf{v}_{S\_P}$ is the vector from the sensor coordinate system to the target position of virtual element P.

Error in Tracker to Sensor Transformation

In the tracker to sensor transformation step, the pose of the head or the HMD is measured frequently and further transformed into the final image. Therefore, in this step, there are a variety of error sources. First, there are many delay-induced errors. They are not further discussed in this thesis since they rely on the specific setup. Second, it includes the tracker-measurement errors named static field distortion, jitter, and dynamic tracker error. The static field distortion is a systematic error like distortions of the sensors. The dynamic tracker error is the error due to movements of the tracker sensor. In a car setting, this can be the shaking of the cameras while driving. Finally, the jitter is an error that cannot be calibrated out of the system, like noise in the pose estimation by the tracker. The jitter is related to the pose estimation. The higher the jitter is, the lower is the precision of the pose estimation system. The overall tracker measurement error can be described as follows:

$$e_{tracker} = ||\delta \mathbf{v}_{T\_S}|| + 2 \cdot \sin \frac{|\delta \varnothing_{T\_S}|}{2} \cdot (||\mathbf{v}_{S\_P}||) \tag{3.2}$$

This shows that an error in translation $\mathbf{v}_{T\_S}$ adds to the overall registration error, but the error in rotation $\delta \varnothing_{T\_S}$ is multiplied by the distance $\mathbf{v}_{S\_P}$ from the sensor S to the point P.

3.1.3 Display Error

Display errors are made in displaying the computed final image to be shown on the HMD display based on the virtual 3D content. Examples of this category's errors can be optical distortion, miscalibration between the virtual image and initial sensor, or aliasing. The most significant error among them is the optical distortion. This error is introduced by the AR glasses and their optics used to render the virtual elements.

3.1.4 Viewing Error

The viewing error contains an error regarding the location of the user's eyepoints in the world coordinate system. The errors contributing to the viewing error can be a calibration error, the rotation of the user's eyes, or slippage of the AR glasses on the user's head. Especially the last error source is to be expected in the driving use case. The overall viewing error can be formulated as follows:

$$s_{view} = e \cdot \frac{|z|}{d} \tag{3.3}$$

where e is the magnitude of the viewing error, d is the distance from the modeled eyepoint to the screen image, and z is the distance from the screen image (or projection plane) to the projected position of the virtual element P'. Most errors contributing to the viewing error are either one time calibration related errors or dependent on the specific HMD.

3.1.5 Relevant Errors for AR Systems in Car Setups

The introduced errors that contribute to an overall AR system can mostly be calibrated and corrected through measurements and are partially negligible, especially regarding our specific car setup and use case. For example, the acquisition error is not applicable, as the objects shown in the AR glasses are fully virtually generated.

Regarding the errors playing into the head-tracking error, some are relevant in our case. The error in world to tracker transformation equals the calibration of the tracking cameras to the world coordinate system. This error is not further targeted, as it is done once in the initial phase, and the calibration is easily definable by traditional camera calibration methods.
The tracker to sensor transformation generally is a crucial error source that we target specifically in this thesis. This error signifies the measured error of the head or the AR glasses pose, being the most decisive error for the overall registration error. The estimation of the 6-DoF pose of the AR glasses or the head by the tracker is done repeatedly. There is a continuous head movement that has to be estimated as accurately as possible. It is fundamental to regress this with high accuracy for a good superimposition of virtual elements and the real world. Some additional errors contribute to the tracker to sensor transformation. One is the mentioned static field distortion, which is usually calibrated out with an intrinsic calibration of the sensors. The dynamic tracker error is bound to external factors like car movement and road conditions. This error can only be minimized by limiting the potential movement of the tracker by firmly attaching the used cameras. In addition, minimizing the jitter is also implicitly part of this thesis. Reducing the jitter of the tracking systems implies a higher precision in the pose estimation of the AR glasses and the head, thus improving the overall AR experience.

Display errors and viewing errors are also negligible, as they are not part of the pose estimation-induced errors in the overall system.

Consequently, the most critical errors that this work aims to reduce are part of the tracker to sensor transformation error. Subsequently, the jitter is automatically reduced with a more accurate and precise tracker to sensor estimation. This estimation coincides with the AR glasses or head pose estimation targeted in this thesis.

3.2 Error Ranges of Car Use Cases

To target the previously introduced errors and reduce them, a quantification of the error is required first. For this purpose, we utilize the following equation, which has been derived based on the equation for the tracker to sensor transformation by Holloway [75]:

$$e_{reg} = ||e_{position}|| + 2 \times ||\sin\left(\frac{e_{orientation}}{2}\right)|| \times d \tag{3.4}$$

We define different variable names for improved readability. e_{reg} is the overall registration error. The registration error is composed of the positional error $e_{position}$, the orientation error $e_{orientation}$, and the distance d to the virtual position of the object. Figure 3.2 visualizes this composition in the case of a person wearing smart glasses while driving.

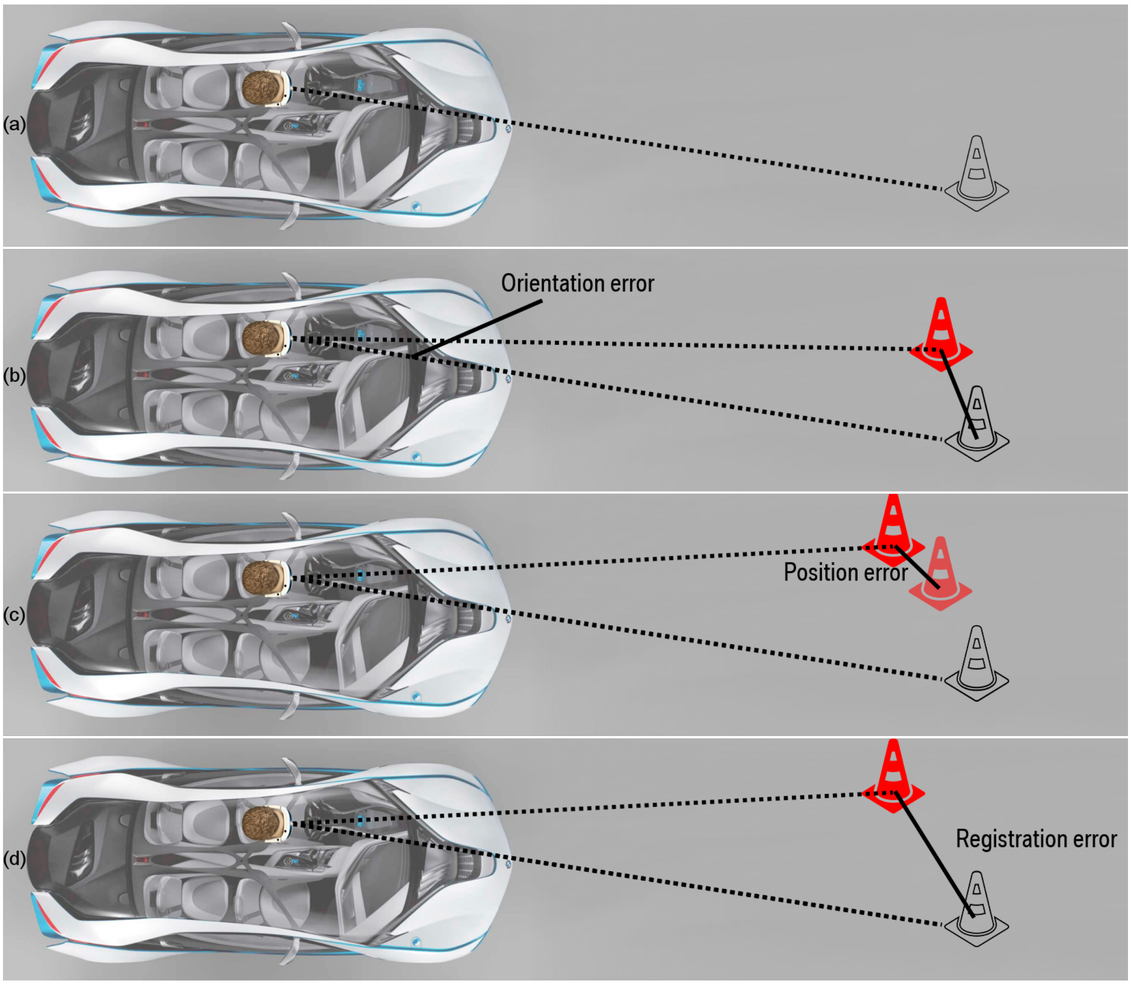


Figure 3.2: Overview of the registration error in our setup. (a) shows where the location of the virtual pylon ought to be. (b) adds an orientation error of the tracking system, resulting in a higher registration error. (c) adds further positional error, resulting in an overall registration error (d).

Based on Equation 3.4, it is possible to compute a maximum error regarding the orientation and position error for certain distances and use cases. The equation shows that a higher orientation error affects the registration error stronger. Thus, the target definition and quantification are focused on the orientation. By setting the positional error target to 5 millimeters, we aim at a positional error below 5 millimeters.

The required maximum error is heavily dependent on the specific use case and its distance from the viewer. Consequently, a definition of categories depending on the use case and the projection distance is useful. The use case in a car setup is dividable by their projection distance into three categories: car interior, car environment, and distant object. Figure 3.3 depicts the three categories, including potential use cases.

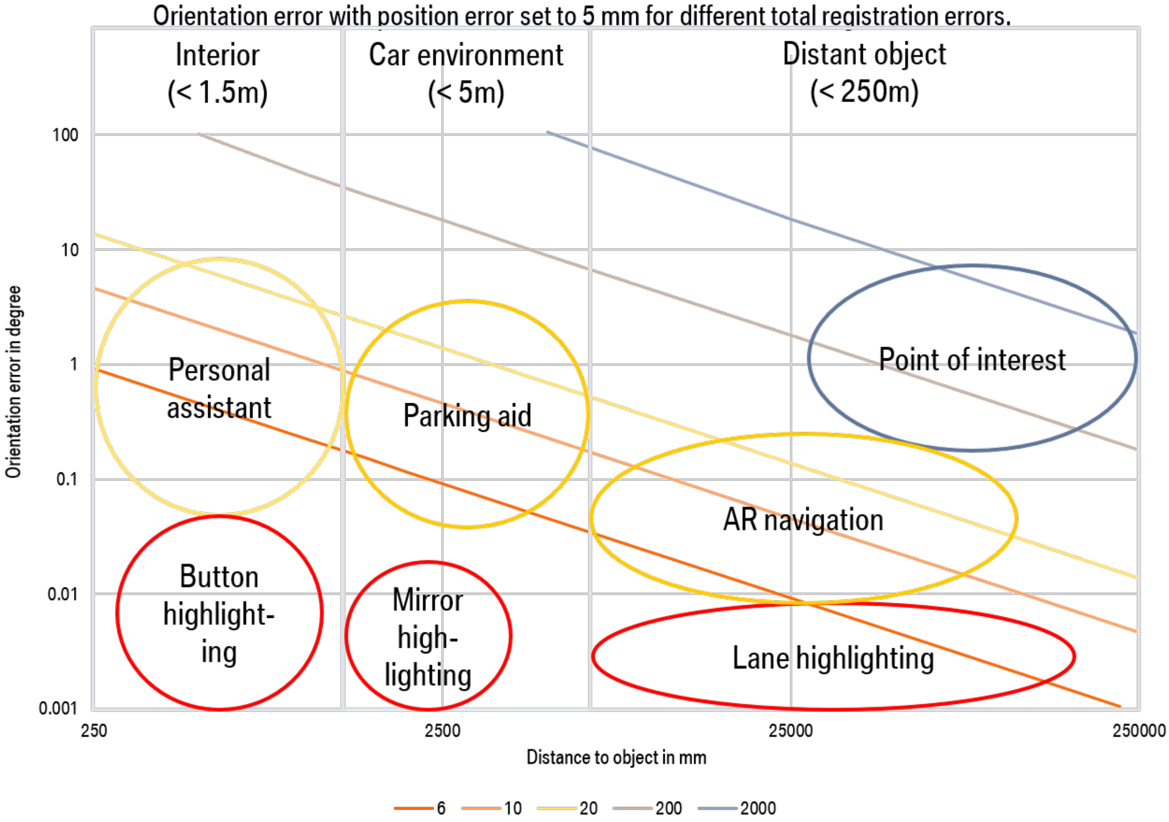


Figure 3.3: Overview of example use cases and their category. The x axis shows the distances and the range of the categories. The y axis plots the orientation in degree. The diagonal lines represent the course registration error when set to 6, 10, 20, 200, and 2000 millimeters. The colors of the lines and circles depict the level of difficulty. They range from red for difficult use cases and challenging registration errors, to yellow to blue.

3.2.1 Car Interior

Use cases in the car interior can be visualized at a 1.5-meter distance from the viewer. In Figure 3.3, two examples of this category are listed: A virtual personal assistant floating over the dashboard and a button highlighting for attention guiding.

A personal assistant (avatar) does not have an expected place by the viewer in the real environment. Hence, for a decent virtual illusion, higher errors are acceptable. In this case, the registration error can be defined to be about 10mm. Hence, the orientation error must be lower than 10 degrees.

A more critical interior use case regarding the potential errors of the tracking system is button highlighting. In this case, hardware buttons can be highlighted and circled for guiding the driver to activate specific functions. This application requires a low total registration error. It must be below the lowest diagonal for registration error in Figure 3.3, which is 6mm. Thus, a maximum error of about 0.1 degree for orientation is required.

3.2.2 Car Environment

The car environment category ranges from 1.5 meters to 5 meters. Use cases with visualizations directly around the car fall into this category. Mirror highlighting and parking aid augmentation are two examples of this category.

In a parking aid application, numerous virtual elements to display the distance to surrounding cars and objects can be shown. The shown virtual content can show higher registration error, as the user of the HMD does not expect it to be at an exact real world position. For this case, the registration error can be more than 10 millimeters. Therefore, the orientation error for a parking aid can be around 5 degrees.

A use case in this category highlighting real world elements exists. In a potential user guiding application, where the driver's attention of a left-hand-drive vehicle has to be guided to the right mirror of the car while the mirror is being highlighted for danger, a low registration error is required. The registration error must be below 6 millimeters, leading to a maximum orientation error between 0.01 and 0.1 degrees.

3.2.3 Distant Objects

In the category of distant objects, the virtual content can be between 5 meters and 250 meters. This includes all use cases placed in the world, which are further away from the car. This category's three different car use cases are mentioned in Figure 3.3: Point of interest, AR navigation, and lane highlighting.

In the case of point of interest highlighting, symbols for famous sights in the surrounding of the city can be shown in the sky. This requires a general direction in the horizon only, why the registration error can be up to 2 meters, and hence result in a maximum orientation error of around 10 degrees.

AR navigation while driving shows arrows and route information located in the real world. This use case is generally located closer to the car in comparison to the point of interest. In addition, the 3D navigation arrows on the road must show a relatively low discrepancy from the lane boundaries and have to lie within them. Subsequently, for a realistic AR navigation, the registration error must be around 20 to 30 millimeters. The orientation error for this has to be below 0.5 degrees.

A difficult use case of this category is lane highlighting. In lane highlighting, the lane boundaries are being highlighted as far as the driver can perceive. For this purpose, a low registration error is necessary, being less than 10 millimeters. This results in a required maximum orientation error of the tracking system below 0.1 degrees.

3.3 Summary

In this chapter, errors that exist in AR systems when using see-through HMDs were introduced first. The most important error sources introduced revolving pose estimation were part of the tracker to sensor transformation. All mentioned errors contribute to the overall registration error, which measures the difference between the target 3D location of a virtual object and its position after rendering. Afterwards, the crucial errors mainly contributing to the registration error in a car setup have been presented, which can be influenced by improvements in tracking the HMD or the head pose.

Subsequently, categories for care use cases have been determined to quantify possible orientation and position errors required for specific use cases. We defined several use cases depending on the projection distance and the importance of a low overall registration error.

The succeeding Chapters 4 and 5 will present several AR glasses and head pose estimation algorithms for in-car deployment. They aim to regress the pose accurately to fulfill as many use case requirements as possible.

4 Deep Learning-based AR Glasses Pose Estimation

As introduced in Chapter 2, a variety of object and head pose estimation methods have been developed in recent years in the Computer Vision community. In terms of outside-in HMD pose estimation, there is a lack of research. Thus, in this work, we propose various Deep Learning-based pose estimation methods for this purpose.

In Chapter 6, we will present the acquired AR glasses and head dataset used to train our algorithms. The dataset consists of multi-view infrared images of car passengers, suitable to experiment with our designed algorithms which can work with multiple images. This chapter presents the designed methods. The methods range from image-based feed-forward pose estimation methods to image-based recurrent pose tracking methods to point cloud-based pose estimation algorithms (Figure 4.1).

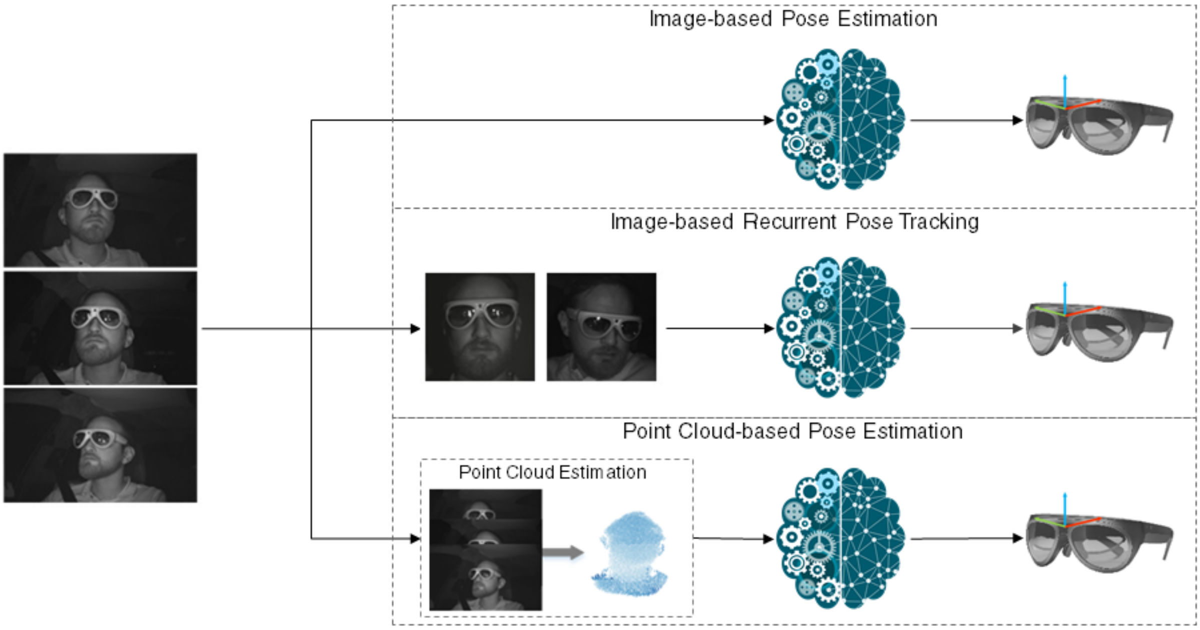


Image-based Pose Estimation
Image-based Recurrent Pose Tracking
Point Cloud-based Pose Estimation
Point Cloud Estimation

Figure 4.1: Overview of the different methods presented in this chapter. Based on infrared images, image-based feed-forward pose estimation methods, image-based recurrent pose tracking methods, and point cloud-based pose estimation algorithms are introduced.

We present image-based feed-forward algorithms first, which constitute the most frequently used form of pose estimation regarding object or head pose. It is followed by our novel recurrent pose tracking methods. The aim behind exploring recurrent pose tracking methods is to potentially exploit temporal information in videos. Then, we propose point cloud-based methods to study the potential of depth information for AR glasses pose estimation. The chapter finishes with fusion methods to fuse several AR glasses pose estimators with the goal of improving the overall pose regression performance.

4.1 Image-based Feed-Forward Pose Estimation

In terms of image-based feed-forward pose estimation algorithms, various methods exist, as presented in Section 2.3. We developed and implemented several image-based AR glasses pose estimation in this work. The first approach is the GlassPose network.

4.1.1 GlassPose Network

GlassPose (Figure 4.2) is inspired by VGG [38] and the POSEidon [71] network.

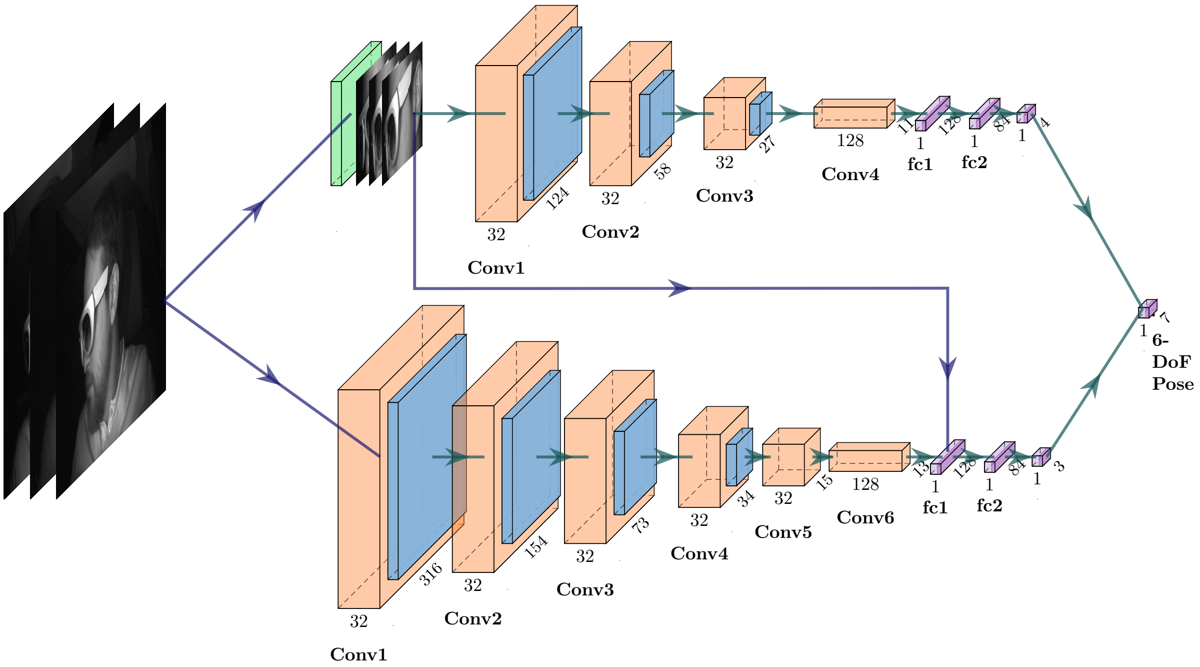


Figure 4.2: The GlassPose algorithm to regress 6-DoF HMD pose. The green square depicts the automated glasses cropping. The upper subnetwork is trained for orientation estimation, the lower subnetwork learns the translation. The translation part contains two more convolutional layers to handle the images with higher resolutions.

Hence, GlassPose performs cropping for the orientation as done in the POSEidon model. It includes two subnetworks, where one is used to estimate to orientation, while the other estimates the translation based on full images.

First, we perform AR glasses cropping by utilizing an existing DNN-based face detector [77] for the orientation subnetwork. The initial full image resolution of one frame is 1280×752. The output of the face detector is adjusted to automatically scale the output to the glasses area with a resolution of 128×54 pixel. The goal of the cropping is reduce the pixel information that doesn't contribute to the orientation estimation. The translation subnetwork expects the full images as input, which are downscaled to a resolution of 320×188 for computational reasons. A cropping is counterproductive to translation estimation as the relative distance to other pixels in the image is useful in that case. The higher input resolution requires more convolutional layers. The first layers of the orientation subnetwork of the architecture are two convolutional layers with a filter size of 5×5, both followed by a max pooling layer. The translation subnetwork adds one more layer with the same specification. For orientation, there are two more convolutional layers with a filter size of 3×3 filter, where a max pooling layer follows the first one. For translation, we add one more convolutional layer without max pooling. Three fully connected layers finalize both subnetworks. During training, the first two fully connected layers use dropout as regularization ($\sigma = 0.5$). We then concatenate the output of both networks to obtain the full 6-DoF pose. The 2D bounding box coordinates generated while cropping the image for the orientation subnetwork are further used in the translation part. The aim is to enhance translation prediction accuracy by appending the normalized bounding box coordinates to the first fully connected layer of the translation estimation subnetwork. The hidden layers deploy the ReLU activation function, while the output layer uses linear activation. As GlassPose is trained similarly to GlassPoseRN, we give further detail on the training in the next subsection.

4.1.2 GlassPoseRN Network

In addition, we present GlassPoseRN, which is based on a ResNet-18 [3] backbone (Figure 4.3). The ResNet backbone consists of several residual blocks with residual connections from previous layers to later layers. This is used to reduce the vanishing gradient problem in CNNs. At the end of the ResNet backbone, we add three fully connected layers with the dimensions 256, 64, and 7 to regress the orientation and translation. The fully connected hidden layers again utilize the ReLU activation function, while the output layer uses linear activation. ResNet is deployed for our task due to its successful application in various Computer Vision tasks in recent years [78, 79, 80].

We train GlassPose and GlassPoseRN similarly. Both networks are trained with an Adam optimizer and an initial learning rate of $\alpha = 0.0001$. The Euclidean distance for translation and orientation based on the loss introduced by Kedall et al. [62] is used for training. Accordingly, the loss function is defined as follows:

$$loss := \beta \left\| t - \widetilde{t} \right\|_2 + \left\| q - \frac{\widetilde{q}}{\|\widetilde{q}\|} \right\|_2 \tag{4.1}$$

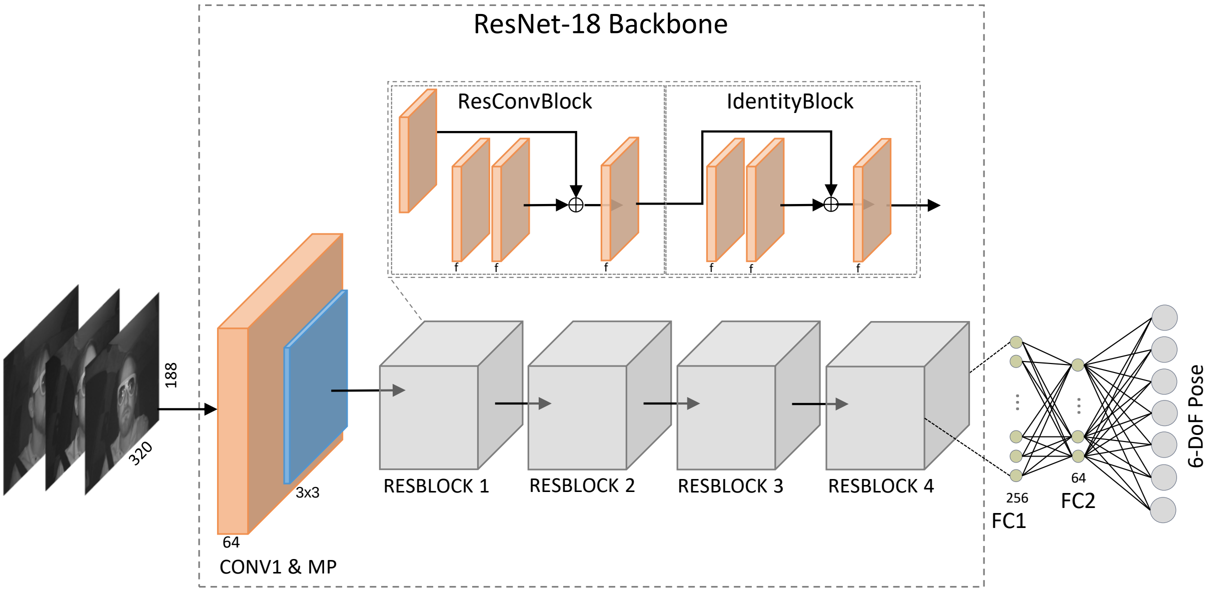


Figure 4.3: An overview of the GlassPoseRN architecture. The ResNet-18 backbone is highlighted with its Residual blocks, containing of residual convolution and identity blocks. In the end of the network, there are the appended fully connected layers to estimate the 6-DoF pose.

q and $\widetilde{q}$ describe the ground truth and the estimated quaternion, respectively. t and $\widetilde{t}$ are defined equivalently for translation. We normalize the predicted quaternion first. Then, the Euclidean distance to the ground truth quaternion is computed. Unit quaternions are regressed on the positive w scale to obtain unambiguous estimations for the orientation. In addition, we compute the Euclidean distance translation. The translation is weighted accordingly through the scaling factor β to achieve a similar scaling to the orientation before being added to the orientation loss. The scaling factor is set to 0.5 to achieve similar scaling levels. Both networks are trained until convergence. The GlassPose networks train for 180 epochs with a batch size of 128, the GlassPoseRN networks for 120 epochs with a batch size of 64. We trained the models on Nvidia Geforce GTX 1080 Ti GPUs.

4.1.3 CapsPose Network

In addition to the previously introduced methods, we propose a novel algorithm for 6-DoF pose estimation based on Capsules. Before presenting the overall architecture, we introduce two key parts of the network: the Capsules and Dynamic Routing.

Capsule Networks

Despite their outstanding success in image-based tasks, CNNs suffer from certain limitations. One property that makes CNNs successful is the translation invariance. To achieve this, the location information is dropped, negatively affecting the generalization performance of pose estimators. A novel network architecture called Capsule Network (CapsNet) [4] was designed to tackle this issue by introducing the equivariance property (Figure 4.4).

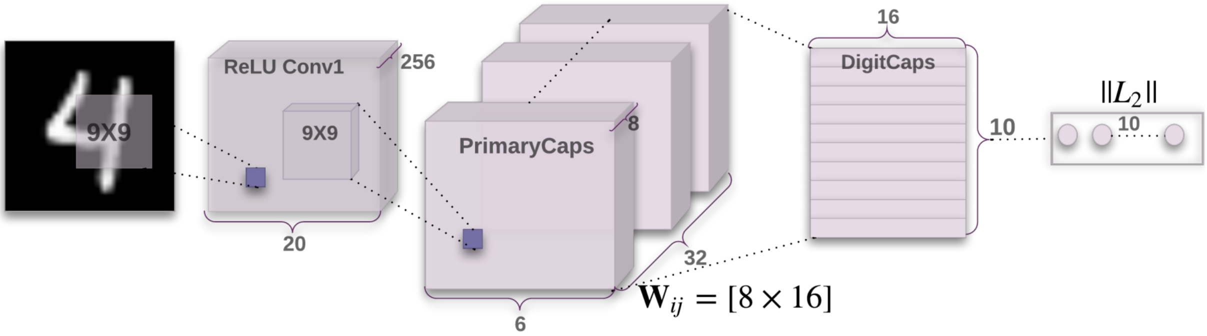


Figure 4.4: The original CapsNet architecture [4].

This property is a generalized form of invariance, in which transformations change the predictability of the model. A CapsNet represents a novel Deep Learning approach that is created to address the weaknesses of traditional CNNs. The elemental entities of CapsNet are called Capsules, which represent a small group of neurons that encapsulate their outputs in an activity vector. This vector represents the instantiation parameters of specific entity types, such as objects or object parts. An output of the neuron within a Capsule corresponds to a property of an entity. Instantiation parameters include properties such as pose, texture, or deformation. A visual system learns the existence of an object based on its relevant features and the relationship between those features and the high-level features. Thus, a process for assigning parts to the whole is needed. For instance, the system must be aware of the relative position of the nose as a part of the face. The association between the simple entity represented by a low-level Capsule, the complex entity represented by a high-level Capsule, and the instantiation parameters of the Capsules are obtained through the routing algorithm. Low-level Capsules represent simple entities that contribute to constructing high-level Capsules of complex shapes. CapsNet [4] consists of a convolutional layer used to extract low-level features. Multiple convolutional operators are used in parallel to construct the first Capsule layer, named the Primary Capsule layer (PrimCaps), followed by the Digit Capsule layer (DigitCaps) consisting of 10 Digit Capsules, each representing the existence of a single digit. The digit existence probability is encoded in the magnitude of the activity vector of the Digit Capsule.

CapsNet was originally designed for classification tasks. Subsequently, we use the architecture as a foundation for our network called Capsule Pose (CapsPose), which adapts it for 6-DoF pose regression. The network is the first to solve the 6-DoF pose estimation problem using Capsules to the best of our knowledge.

Dynamic Routing

Dynamic Routing is critical to connecting low-level, primary capsules with high-level Capsules. Given an input image, low-level features are extracted first. This results in building the low-level primary Capsules. Two low-level Capsules represent the mouth and the nose for example. The local position of these features is described by the activity vector of the corresponding Capsules. A trainable transformation matrix transforms the position of the features from their local coordinate systems to the coordinate system of the higher-level Capsule, which is the pose Capsule. According to the coupling coefficients, its activity vector is computed. It embodies the position of the target object represented in its own Capsule space. A capsule projector maps the activity vector from the capsule space into the target pose space.

CapsPose Network Architecture

Figure 4.5 shows an overview of the CapsPose architecture.

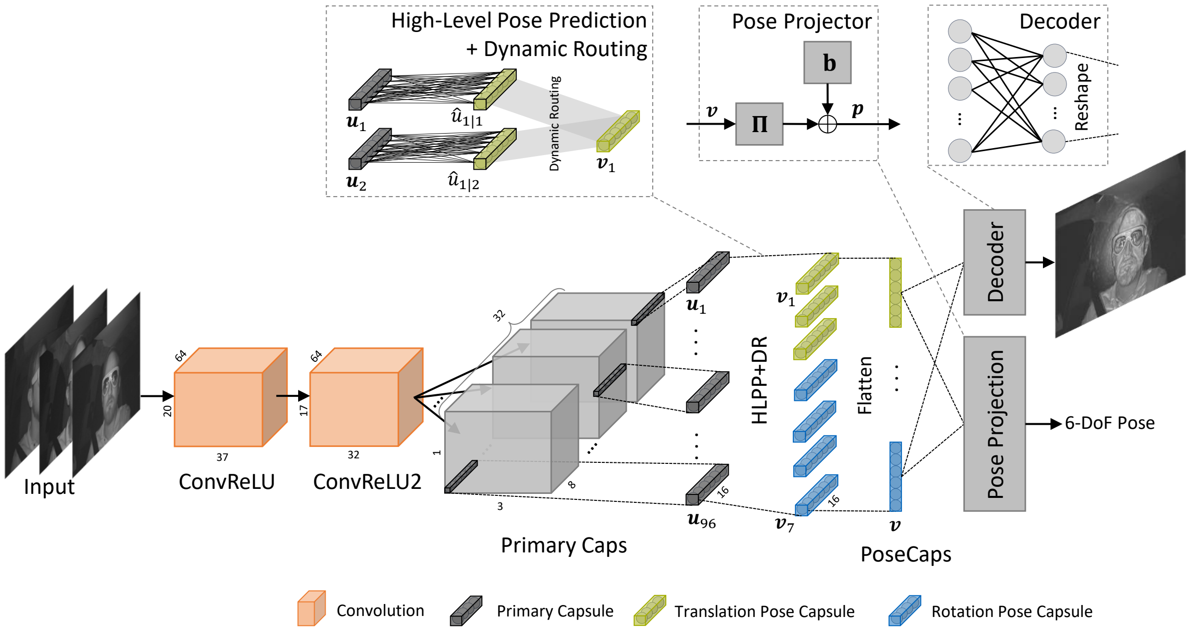


Figure 4.5: Our CapsPose architecture. The ConvReLU layer extracts low-level features. The parallel convolutional layers form the primary Capsules representing low-level entities. A high-level pose prediction step and the Dynamic Routing algorithm transmit the information to the pose Capsules. A pose projector projects the activity vector of pose Capsules onto the pose space. A Decoder reconstructs one image from the triple input images.

CapsPose consists of 4 layers: two convolutional layers called ConvReLU and ConvReLU2, PrimaryCaps, and an output Capsule layer called PoseCaps. The ConvReLU layer detects basic features in the input. It inputs down-sampled triple images in the form of inputs tensor of size $320 \times 188 \times 3$ pixels. The given pixel data is converted to the activity of local feature detectors leading to the extraction of low-level feature maps. The layer ConvReLU generates 64 feature maps over 32×32 filters with a stride of 8×8, and no padding. The second layer is also a convolutional layer with 64 filters of size 8×16 and a unit stride. We use a 2×5 padding to obtain a feature map of size 17×32. Both layers use $ReLU$ function as their activation function. The third layer is a convolutional Capsule layer. This layer can be viewed as a convolutional layer with 256 filters of size 16×16, 8×8 stride and no padding. Rearranging the result produces 96 8-dimensional primary Capsules. The squashing function [4] activates this layer. The next layer is the PoseCaps layer, which consists of 7 Capsules including 16 instantiation parameters. In the following, the translation is presented in the Euclidean space, the rotation in the quaternion space leading to a pose space of 7 dimensions. Each pose Capsule receives the 8-D input of all 96 primary Capsules. Each of these input vectors receives its own 8×16 weight matrix, which maps the 8-D input space to the 16-D Capsule output space.

The routing-by-agreement takes place only between the PrimaryCaps and PoseCaps layers. The local position of these features in the PrimaryCaps is described by the activity vector of the corresponding Capsules. The dynamic routing and, thus, the routing-by-agreement between the PrimaryCaps and PoseCaps defines a trainable transformation matrix. It transforms the orientation and position of the features from their local coordinate systems to the coordinate system of the higher-level Capsule, which is the pose Capsule. According to the coupling coefficients, its activity vector is computed. It embodies the pose of the target object represented in its own Capsule space. We add a projector module, which maps elements from the instantiation parameter space of the Capsules to the 7-dimensional pose space. The following equation describes the projection operation:

$$p = \Pi v + b, \tag{4.2}$$

where v is a vector stacking the activity vectors of all Capsules v_j with $j = 1, \ldots, 7$. The pose vector p contains predicted pose values consisting of the translation and rotation. Since the positions and the orientations are being estimated as two decoupled tasks, we use the following projection matrix structure:

$$\Pi = \begin{bmatrix} \pi_{1,1}^{\top} & 0 & 0 & & & \\ 0 & \pi_{2,2}^{\top} & 0 & & \mathbf{0} & \\ 0 & 0 & \pi_{3,3}^{\top} & & & \\ & & & \pi_{4,4}^{\top} & \cdots & \pi_{4,7}^{\top} \\ & \mathbf{0} & & \vdots & \ddots & \vdots \\ & & & \pi_{7,4}^{\top} & \cdots & \pi_{7,7}^{\top} \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_7 \end{bmatrix}, \boldsymbol{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_7 \end{bmatrix}, \tag{4.3}$$

$\pi_{i,j}$ projects the pose Capsule i onto the j-th dimension of the target pose. The axes are decoupled of the translations since the dimensions of the Euclidean space are uncorrelated. However, the projector uses the activity vectors of the Capsules 4 to 7 jointly to obtain the quaternion values. By this, we improve the projector's exploration of the quaternion space, as relationships between the dimensions of the quaternion 4D space exist (e.g. normalization constraint and symmetry of the hypersphere).

We add a reconstruction or decoding module to enhance the feature learning, which reproduces one channel of the input image given the values of all activity vectors stacked in v. The vector v is considered as a code that encapsulates all vital information for the pose prediction and the reconstruction. This additional network is inspired by Sabour et al. [4] and uses their reconstruction loss. The decoder consists of two fully connected *ReLU* layers with 512 and 1024 neurons and one fully connected *Sigmoid* output layer. This part of the network is used to improve the quality of the learned features only.

For training, the Adam optimizer with the learning rate $\alpha = 0.0001$ is used for 180 epochs with a batch size of 128. We use the same loss function as for the previous methods. CapsPose was trained on Nvidia Geforce GTX 2080 Ti GPUs.

4.1.4 Benchmarked State-of-the-Art Image-based Method

In order to compare our novel methods to the state of the art, we select a recent image-based direct pose estimation method based on classification [81]. Berg et al. implement and evaluate various techniques for discrete class representation creation for regression via classification (RvC). They propose the usage of several discrete data representations simultaneously to improve neural network learning. Regarding head pose estimation, the most promising discrete class representation method is ”RandomizedBins”. In this case, they introduce a set amount of class intervals D^m in contrast to traditional RvC approaches with one class interval. Within each interval, they randomly sample L bins of varying width to maximize diversity between different discretizations D^m. Therefore, target values that do not belong to any of the chosen classes are assigned to the nearest neighbor in the sample. They utilize ResNet-50 [3] as a backbone with additional fully connected layers for each estimated value. They use M fully connected layers for each class D^m and L fully connected layers per class with a softmax layer for all outputs. Figure 4.6 shows the overall architecture.

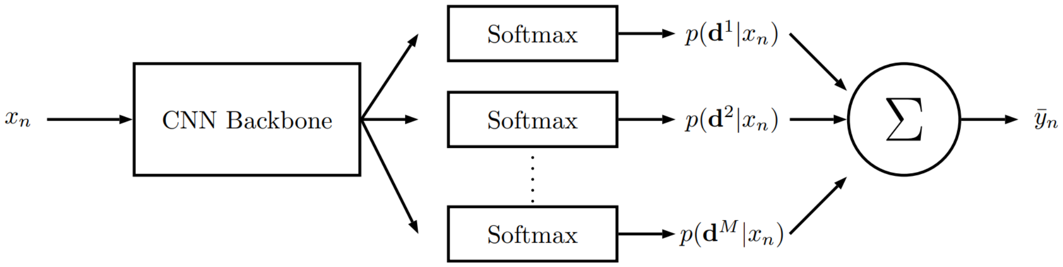


Figure 4.6: Overview of the RvC-based pose estimation method [81]. $p(\mathrm{d}^M|x_n)$ depicts the softmax layer outputs per class D^m for an input x_n. The outputs are being combined using an ensemble average to obtain a final pose estimation.

The original implementation is focused on orientation estimation only. We adjusted the architecture to enable full 6-DoF pose estimation. We realize this by introducing discretizations for the position equivalent to the orientation. Therefore, the values for classes of six different outputs are estimated instead of the initial three outputs for orientation.

The method is also trained on the generated HMD dataset, which will be presented in Chapter 6. The parameters for the method are set identical to the original implementation. Thus, L is set to 40 random bins per class M, which is set to $M = 30$. They are randomly sampled by dividing the label ground truth range in steps of 0.01 degree and 0.01 cm. The Adam optimizer is used with an initial learning rate of $\alpha = 0.0001$. We trained the model on Nvidia Geforce GTX 1080 Ti GPUs. In the remainder of the work, this approach is referred to as ”RandomizedBins”.

4.2 Image-based Recurrent Pose Tracking

As an alternative to the models mentioned above, this section introduces RNN-based networks to compare the convolution and recurrent operations. Their main aim is to exploit the temporal information in the data by introducing RNNs in the architecture. CNNs can learn spatial information avoiding feature engineering, while RNNs avoid manual engineering of object tracking logic, which can be prone to errors. They learn the temporal information directly from the data. Hence, a hybrid model could achieve enhanced AR glasses pose estimation.

LSTMs are used as the RNN variant due to their ability to avoid the vanishing gradient problem and the poses time-dependent property. The LSTMs map the pose estimation per frame in a sequence to the sequence of known ground truth poses. First, a CNN backbone for feature extraction is required.

4.2.1 BaselineCNN - Backbone for RNN methods

First, we design a CNN for the LSTM models to build upon, called ”BaselineCNN”. It is a simple CNN and is inspired by the ”GlassPose” method. Instead of decoupling the network into subnetworks for translation and rotation, only one network estimates 6-DoF pose estimation at once for simplicity. BaselineCNN neither uses any extra cropped images of glasses nor the bounding box of glasses in the translation subnetwork. It utilizes full downscaled images to regress the 6-DoF pose per frame triple. The network learns the rotations in quaternion space and translation in Euclidean space.

The network comprises six layers in total. The input to the network is of size 320×188 pixels. Initially, three convolutional layers of filter size 5×5 process the images, each layer having 32 neurons followed by a max pooling layer of 2×2. The remaining three convolutional layers have a filter size of 3×3. The fourth and fifth convolution layers have a hidden size of 32, while the sixth convolution layer has 128 neurons. Only the fourth layer has a consecutive max pooling layer.

We also train this network without subsequent LSTM layers for later comparison. In this case, fully connected layers of sizes 128, 80, and 7 neurons are added to regress the 6-DoF pose. The model trains with an Adam optimizer with the initial learning rate $\alpha = 0.001$. The learning rate is scheduled to decrease if the validation loss has not improved for more than ten epochs. We trained the models on Nvidia Geforce GTX 2080 Ti GPUs.

4.2.2 LSTM-based Approaches

In this subsection, we describe three different models for 6-DoF pose tracking. They utilize either the BaselineCNN, or the GlassPoseRN as a backbone. A variant of the BaselineCNN-

LSTM combination introduces separable convolutions for potential improvement in estimation accuracy and efficiency.

CNN-LSTM

The first model is referred to as ”CNN-LSTM”. The CNN part of the network is identical to the BaselineCNN backbone described above and extracts optimal features from the IR images, and LSTM tracks the extracted features as sequences. Therefore, the features from a single triple image in a sequence act as one time step for LSTM. The designed model takes a sequence of IR images as input and outputs continuous 6-DoF pose of AR glasses, tracking the pose in all the sequence frames.

The architecture consists of the BaselineCNN model with two consecutive LSTM-layers. The features from the CNN are flattened before using it as an input to the LSTMs. The LSTM network comprises two layers, each having 128 neurons. The dense layers are used as the output layers to regress the continuous 6-DoF pose of AR glasses.

SepConv-LSTM

This method introduces a more efficient network by utilizing depthwise separable convolutions instead of regular convolutions. Depthwise separable convolutions [82] are a special type of convolutions in which convolution operations are separately applied to channels. The information from multiple channels in the input is not mixed, which are the different views of the triple images. Pointwise convolutions increase the depth of the output to be further processed. Separable convolutions increase efficiency by decreasing the number of computations. Standard architectures like MobileNet [83] also use them.

Figure 4.7 shows an overview of the architecture, referred to as ”SepConv-LSTM”. The network architecture remains the same as CNN-LSTM. We only replace regular 2D convolution layers with separable convolutions. Table 4.1 highlights the difference in parameters, showing the reduction of approximately 40,000 parameters using separable convolutions.

Network	Total Parameters
CNN-LSTM	3,849,575
SepConv-LSTM	3,808,274

Table 4.1: Size and parameter comparison of CNN-LSTM and SepConv-LSTM.

GlassPoseRN-LSTM

The next LSTM-based model is the GlassPoseRN method coupled with LSTMs. We designed this model for spatio-temporal data. The LSTM at the end of the network learns from the temporal information and is used in a regression manner to predict continuous 6-DoF AR glasses

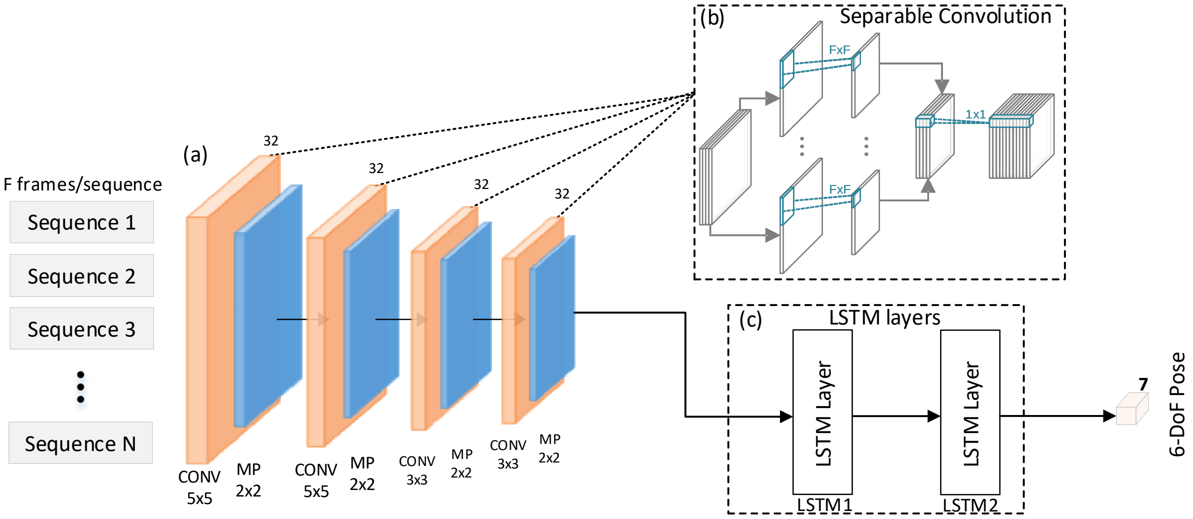


Figure 4.7: The network architecture of separable convolution-based CNN-LSTM model. (a) shows the CNN architecture, including the separable convolutions highlighted in (b), which elaborates on their working principle [82]. The blue layers right after the orange separable convolutions are max pooling layers. In (c), the output of the network is fed into LSTM layers, performing estimation of the orientation in quaternions and position.

pose. We call this network ”GlassPoseRN-LSTM”. The model is built over a ResNet-18 backbone as in GlassPoseRN. This enables comparisons of how the backbone affects HMD pose tracking.

We modify the GlassPoseRN model by removing the three dense layers at the end and adding the LSTM layers. It is done by first adding a global average pooling layer right after the ResNet blocks to reshape the feature maps. Two LSTM layers with 128 neurons each follow mentioned pooling layer. In the end, a dense layer regresses the translation and rotations, making it a layer of 7 neurons.

All networks presented in this subsection were trained identically to the Recurrent Neural Networks in the following subsection regarding RNNs with Non-Local Blocks. Therefore, we give further detail on the training in the next subsection.

4.2.3 RNNs with Non-Local Blocks

The previous subsection used convolutional and recurrent operations to build networks due to the nature of the data modality. This subsection describes a novel neural network architecture based on non-local operations. Non-local operations help in learning long-range dependencies in the video sequence. Wang et al. [84] introduced non-local filtering as general operations in a neural network as a building block to learn the non-local neighborhood information and long-range dependencies. Before introducing this variant, learning the long-range dependencies was done by either repeatedly deep stacking convolutional operations with large receptive fields or deep stacking of recurrent operations consecutively. However, this multi-hop dependency with deep stacked networks is hard to model and difficult to optimize, making it an inefficient and computationally expensive solution.

NL-CNN-LSTM

Our CNN-LSTM model is altered by adapting non-local blocks in the CNN part of the network. The convolution and recurrent layers only consider the local neighborhood of a given pixel position while learning features from the input and the fully connected layers lose the positional correspondence. Hence, the long-range dependencies between the sequenced IR frames and even the non-local neighbor pixels within one frame are missed. Therefore, non-local blocks add the capability to learn this non-local information within a single frame and between multiple frames of a sequence to the network.

We design the network by modifying the CNN-LSTM architecture. The change includes adding non-local blocks in a conventional convolutional network part to experiment with their performance in an LSTM-based network. Figure 4.8 visualizes the architecture.

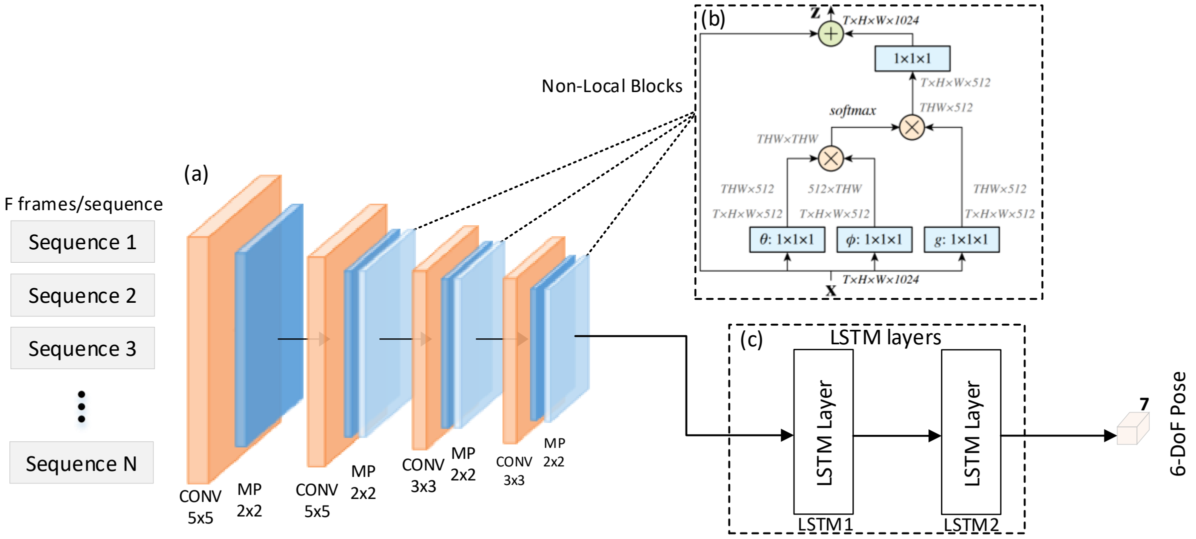


Figure 4.8: The network architecture of non-local block-based CNN-LSTM model. (a) shows the CNN architecture, including the non-local layers highlighted in (b), which elaborates on their working principle [84]. In this block, ⊗ denotes matrix multiplication and ⊕ represents element-wise addition. The light blue box within the non-local block represents the 1x1x1 convolutions. In (c), the output of the network is fed into LSTM layers, performing estimation of the orientation in quaternions and position.

The light blue blocks in the figure represent the location of where the non-local blocks are inserted. Due to limited computation resources and a large image resolution size, the non-local block is not inserted after the first convolutional layer, despite its potential benefit to the overall network. We keep the architecture similar to the CNN-LSTM architecture to ensure a fair comparison between the two models. This network consists of six layers with four convolutional layers, three non-local blocks, and two LSTM layers. The convolutional layers are identical to the CNN part of the CNN-LSTM neural network. The non-local blocks do not affect the size of the input feature maps. The block performs multiple 1×1 convolutions to receive different embeddings. The embeddings are later used in element-wise multiplication and get the final output feature vector, representing the similarity between a pixel and its non-local neighbors. This approach is an extension of the CNN-LSTM model, with the potential to learn the long-range dependencies of different objects for the AR glasses pose estimation task.

NL-SepConv-LSTM

Following the idea of depth-wise separation of convolutions like SepConv-LSTM, we implement a similar model with the additional inclusion of non-local blocks. As non-local blocks are introduced to learn non-local neighbor features [84], we aim to learn the non-local neighbor features from separate views by their inclusion in the network.

The architecture of the network remains the same as NL-CNN-LSTM, which was described in the previous subsection. The only change is the replacement of standard convolution layers by separable convolutions. This network is called ”NL-SepConv-LSTM”.

Training Details

All introduced RNN-based models of this subsection and Subsection 4.2.2 are trained on a similar loss as GlassPose and GlassPoseRN, as presented in Subsection 4.1.2. The only difference is an additional regularization parameter γ and the norm of the predicted quaternion is added in the loss to cater to any large predictions. However, this leads to comparable results as without regularization. Sequences are generated by stacking the frames, sorted according to the timestamp. Our activation function for all layers except for the output layer is the ReLU activation. A linear activation is deployed for the output layers.

We train all networks on Nvidia Geforce GTX 2080 Ti GPUs for 200 epochs on the HMD-Pose images, down-scaled to 320×188 pixels. We set the initial learning rate to $\alpha = 0.001$, which is scheduled to 50% decrease after every ten epochs if the validation loss has not improved from the previous best loss. The batch size used for the BaselineCNN method is 128, while, for RNN-based methods, a sequence length of 8 and batch size of 32 was utilized while training. The trained networks can predict an absolute 6-DoF pose per frame, up to a pose per sequence of 8 frames.

4.2.4 Sequential CapsPose Network

The last LSTM method is the Sequential Capsule Pose Network (SeqCapsPose) model. We extend the CapsPose network by incorporating recurrent blocks into its architecture. Two convolutional layers extract low-level features followed by two Capsule layers learning spatial information (part-whole relationships), yielding a first pose prediction represented in the Capsule space. This Capsule backbone exploits the complete spatial information, learning the spatial features are the relationships between them. Instead of using a non-recurrent projection from the Capsule space to the target pose space, a recurrent block learns the correlation of the Capsule codes in time. A learnable linear function is used to scale the output $y^{(t)}$ as follows:

$$p_i^{(t)} = w_i y_i^{(t)} + b_i, \tag{4.4}$$

where $y_i^{(t)}$ is the i-th value of the recurrent block output vector $y^{(t)}$ at time step t. The variables w_i and b_i are learnable parameters.

The non-recurrent projection consists of two LSTM-layers. They enable learning short and long-term dependencies in the data via the cell states and the hidden states. Figure 4.9 shows the architecture of the SeqCapsPose network.

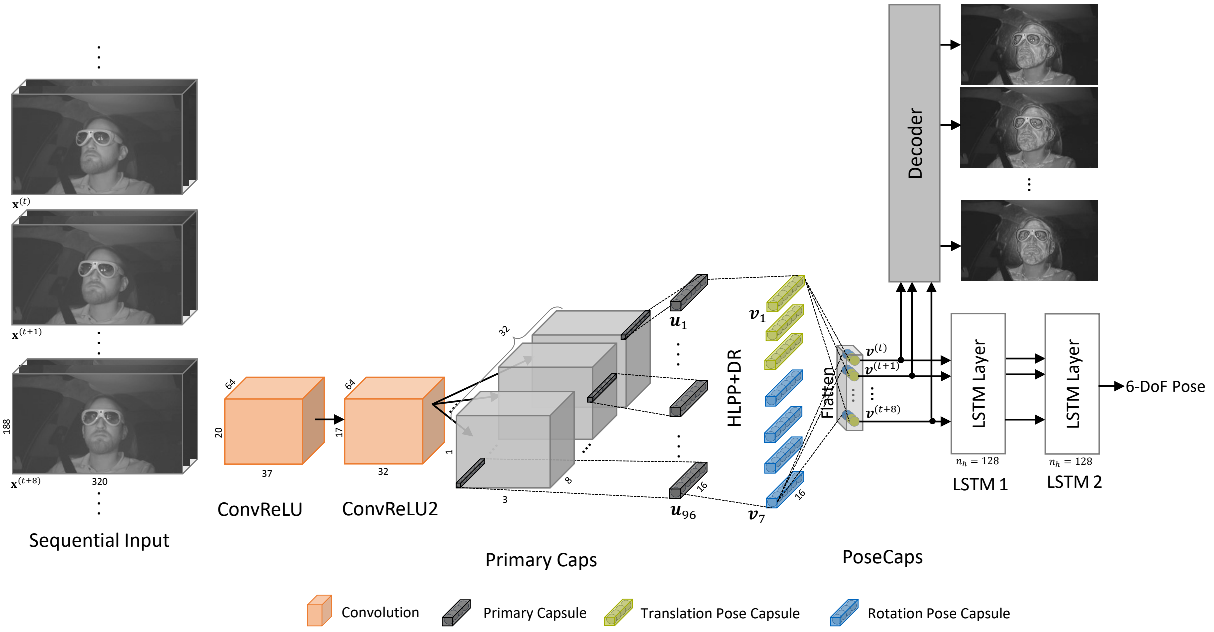


Figure 4.9: The SeqCapsPose network architecture. Sequential images are fed to the network. A time distributed convolutional layer extracts low-level features. Time distributed convolutional layers extract in parallel features to build the primary Capsules. Activity vectors are fed to a recurrent pose projector consisting of two LSTM layers to exploit the temporal information. An image decoder is used to enhance the quality of the learned features.

We train SeqCapsPose similar to CapsPose. An Adam optimizer with the learning rate $\alpha = 0.0001$ is used to train the SeqCapsPose network for 180 epochs, with the same loss function and a batch size of 256. The batches are fed to the Capsule backbone and then redistributed according to their timestamps to form batches of 32 samples with sequences consisting of 8 consecutive frames.

4.3 Point Cloud-based Pose Estimation

Another integral part of Neural network architectures used for 3D object detection and classification as well as scene semantic parsing, are 3D input or point clouds. Since they include geometric information of the target object, they usually lead to more accurate results. It is vital to study them since depth sensors are sometimes used in automotive interiors. Depth information can be also generated from 2D images. For the object pose problem, some approaches estimate the bounding box center for translation and heading for rotation based on point cloud input [85, 86]. The rotation is defined with only a one-dimensional heading angle. However, in this work, the aim is three-dimensional rotation information. The specialized PointNet++ layers [85] have been proven to extract useful information from point clouds in the form of so-called critical points or seed points. From an object classification point of view, these points represent the minimum set of points, still providing enough information to identify an object [5]. Thus, this set is a good descriptor for a point cloud. Based on a PointNet++ framework, we develop and present two deep neural network architectures called PointsToRotation (P2R) and PointsToPose (P2P). The P2R architecture directly regresses 3D rotation from the target object's found seed points (Subsection 4.3.3). The P2P method extends the first method by a voting module and a translation estimation (Subsection 4.3.4). The two methods require point clouds, first estimated with a custom, multi-view trained, monocular point cloud estimator.

4.3.1 Point Clouds Generation

To generate point clouds to train pose estimators, we design a point cloud estimator, which generates point clouds based on the multi-view IR-images of the HMDPose dataset. The dataset will be further presented in Chapter 6. The dataset contains synchronized image triples per pose. The images come from cameras positioned to the left, right, and directly in front of the driver wearing the AR glasses.

The point cloud estimator is a semi-supervised Deep Learning approach. The network learns by projecting the generated point cloud onto other frames of the same scene captured from a different perspective than the input frame. The projection per view is compared to a ground truth foreground mask of this perspective's respective image. By minimizing the offset between projection and ground truth per view, the network can produce a 3D representation that is suitable for all views. The network uses ground truth translation and generated masks for self-and semi-supervision. Figure 4.10 illustrates a full overview of the proposed pipeline, including the losses.

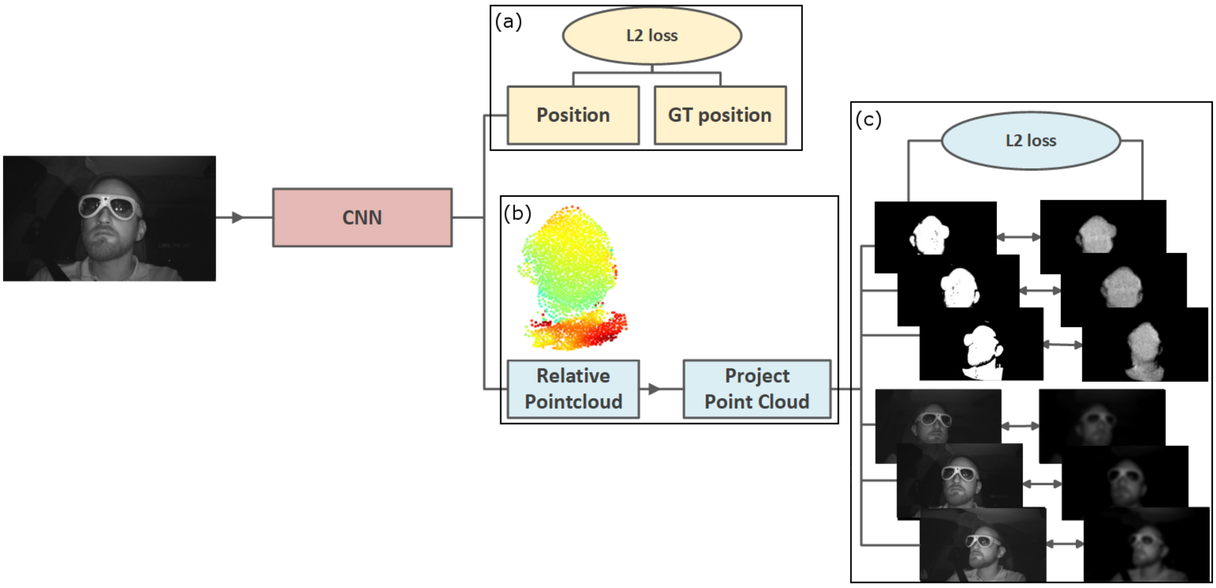


Figure 4.10: An overview of the point cloud estimation architecture. (a) The translation is estimated based on the ground truth. (b) The point cloud estimation is performed with self-supervision based on (c) pregenerated masks of the target objects and the intensities of the original images.

Mask Generation In the first preprocessing step, we generate masks for supervising the point cloud estimation network. The supervision is based on the comparison of the generated masks with the 2D projections of predicted 3D points. The masks assure that the network learns to omit unnecessary background information and focuses on the foreground information from the input images for point cloud generation (Figure 4.11).

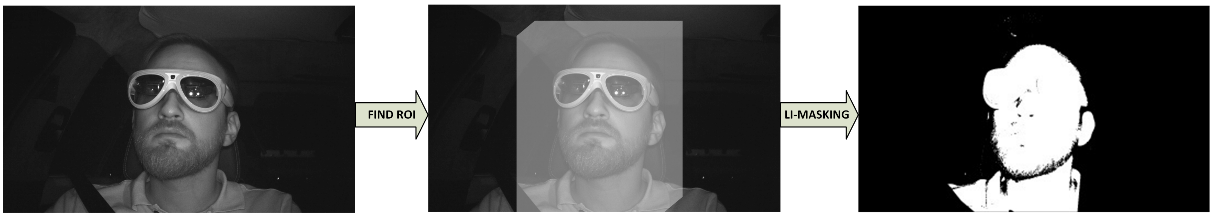


Figure 4.11: Mask generation pipeline. From left to right: determine region of interest based on projection of rendered cuboid and Li-Thresholding.

First, the region of interest in the image is defined by rendering a cuboid based on the current ground truth pose given by the HMDPose dataset. The cuboid approximates the upper body and filters noise visible from the background. The cuboid has the sizes width = 17.8cm, height = 26.4cm and depth = 25.7cm. In the next step, Li-thresholding [87] [88] is applied, which is a method to separate the foreground from the background by iterative cross-entropy minimization. This method is effective due to the foreground being much lighter than the background, and thus the object can be separated easily.

Neural Network for Point Cloud Estimation We propose a Deep Neural Network to estimate the object's intensity, shape, and location in a given input image. A CNN is trained to predict point cloud information in a semi-supervised manner, based on positional ground truth information and multi-view camera images. The predicted point cloud involves positional and intensity information per point. The training on intensity and scale are crucial to stabilize and improve the point cloud quality during training. The network benefits from the intensity and scale information to estimate 3D points on the correct 3D position during training. To assess the accuracy of the predicted point cloud, it is transformed and projected it into the other camera views using an adapted version of the differentiable point cloud renderer proposed by Insafutdinov et al. [89]. Figure 4.12 shows a detailed overview of the utilized CNN.

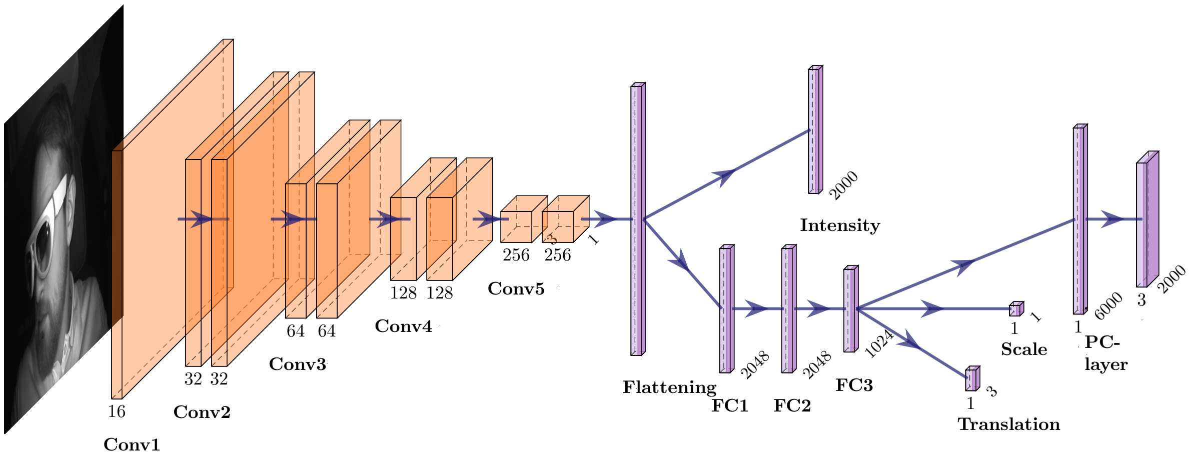


Figure 4.12: An overview of the CNN architecture of the point cloud estimator. Convolutional layers extract features from the input image. We double the number of filters per convolutional block. Kernel size changes between one and three, starting with three in the first layer of the first convolutional block. While color information is directly regressed from the flattened feature vector, a decoder processes the feature vector before scale, translation and point cloud are predicted.

One input image of a given triple is fed through the network. First, the encoder extracts features from the input images. Extracted features are flattened and further used to regress the intensity information per point directly. Having these encodings, three further fully connected layers are used to predict the overall translation of the object t', a relative point cloud of 2000 points with x, y, z coordinates, and a scaling factor s to control the points' size. The intensity information is already present in the input image. This is why this information is directly derived from the convolved features.

The center of the point cloud and its prediction is positioned in the relative center per point cloud, since the predicted 3D points determine the offset to the predicted translation value. This approach enables separating shape and positional information and improve them in separate loss terms. The CNN predicts a point cloud to be projected into the different camera views to be used for the mask projection loss $\mathcal{L}_{mp}$. For this projection the camera intrinsics C and camera extrinsics T are required. We homogenize the predicted points p^s, resulting in the points p^s_h. The ground truth translation t is then added per point. For a given camera among the left, right, and center cameras, the camera coordinates p^c are computed using the inverse camera intrinsics C^{-1}:

$$p^c = C^{-1} * (p^s_h + t) \tag{4.5}$$

Then, the world coordinates p^w can be computed by multiplying them with the inverse camera extrinsics T:

$$p^w = T^{-1} * p^c \tag{4.6}$$

Taking into consideration the ground truth translation and the predicted depth information per point, the image coordinates are computed for the other camera views by projecting p^w to the image plane per view. The remaining projection pipeline follows one of the differentiable point cloud projector proposed in [89].

Three different loss terms are used to improve the predictions of the network. We adapt the loss terms for the mask projection and intensity mask projection from Insafutdinov et al. [89], with the addition of normalization to each term.

- **Translation loss** $\mathcal{L}_t$ We measure the accuracy of the positional prediction as the L_2 loss between ground truth translation and the predicted translation after conversion to the object box. This box's borders are set as the minimum and maximum translation values per axis based on the ground truth annotations. This is visualized in Figure 4.13.

- **Mask projection loss** $\mathcal{L}_{mp}$ The mask projection loss measures the L_2 loss between the predicted point cloud's projection and the generated masks. We smooth the ground truth mask with a Gaussian kernel with filter size $K = 3$ and variance $\Sigma = s$, where s is linearly increased from 0.2 to 1.0 during training. The image resolution normalizes the loss.

- **Intensity mask projection loss** $\mathcal{L}_{cmp}$ This loss term measures the discrepancy between the intensity mask projection and the intensity ground truth information by applying an L_2 loss. Therefore the predicted intensity information is supplemented to the point cloud prediction before projection. Intensity masks are generated by masking the original input images with the generated masks according to Section 4.3.1. The following procedure is similar to the mask projection loss.

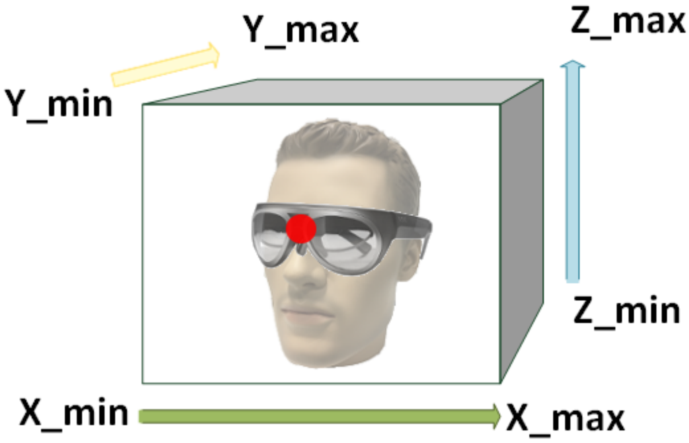


Figure 4.13: Prediction relative to object box. The predicted translation (red) is relative to a pre-defined object box. The borders of this box have been determined as the minimum and maximum translation values per axis based on the ground truth annotations.

We compute the overall loss as a weighted average of the three loss terms:

$$\mathcal{L} = a * \mathcal{L}_t + b * \mathcal{L}_{mp} + c * \mathcal{L}_{cmp} \tag{4.7}$$

a, b and c is for the weighting of the loss terms. The values are set to $a = 0.2$, $b = 0.2$ and $c = 0.6$, which are derived from experiments. This weighting assures that the crucial translation loss $\mathcal{L}_t$ and the mask projection loss $\mathcal{L}_{cmp}$, have more weight in the overall loss. The intensity mask projection loss $\mathcal{L}_{mp}$ assists them in strengthening the point cloud estimation quality. Figure 4.14 shows example results of the point cloud estimation. Each row shows a point cloud estimation of a different person wearing a different type of glasses. The figure displays the point clouds from slightly different perspectives. The point clouds are visualized in the typical color-coding scheme for depth information. Close points to the camera are blue, whereas more distant 3D points are red.

Training Details We rescale all images of the HMDPose dataset to a quarter of their original input size of 1280×752 pixel to 320×188 pixel and normalized for training and testing. The network is trained on Nvidia Geforce GTX 1080 Ti GPUs using Adam Optimizer with a learning rate of 0.0001 and standard momentum parameters. The model trains by randomly selecting one of the three views as input. The predicted point cloud is projected to a voxel grid of $160 \times 94 \times 160$ for width, height, and depth, respectively.

Difference in Training and Testing While the training procedure relies on multi-view input images and ground truth positional information, only one input image of an arbitrary view is needed at test time. During training, the network learns to produce point cloud information based on one input image, the other two views have only been used for semi-supervision. Since

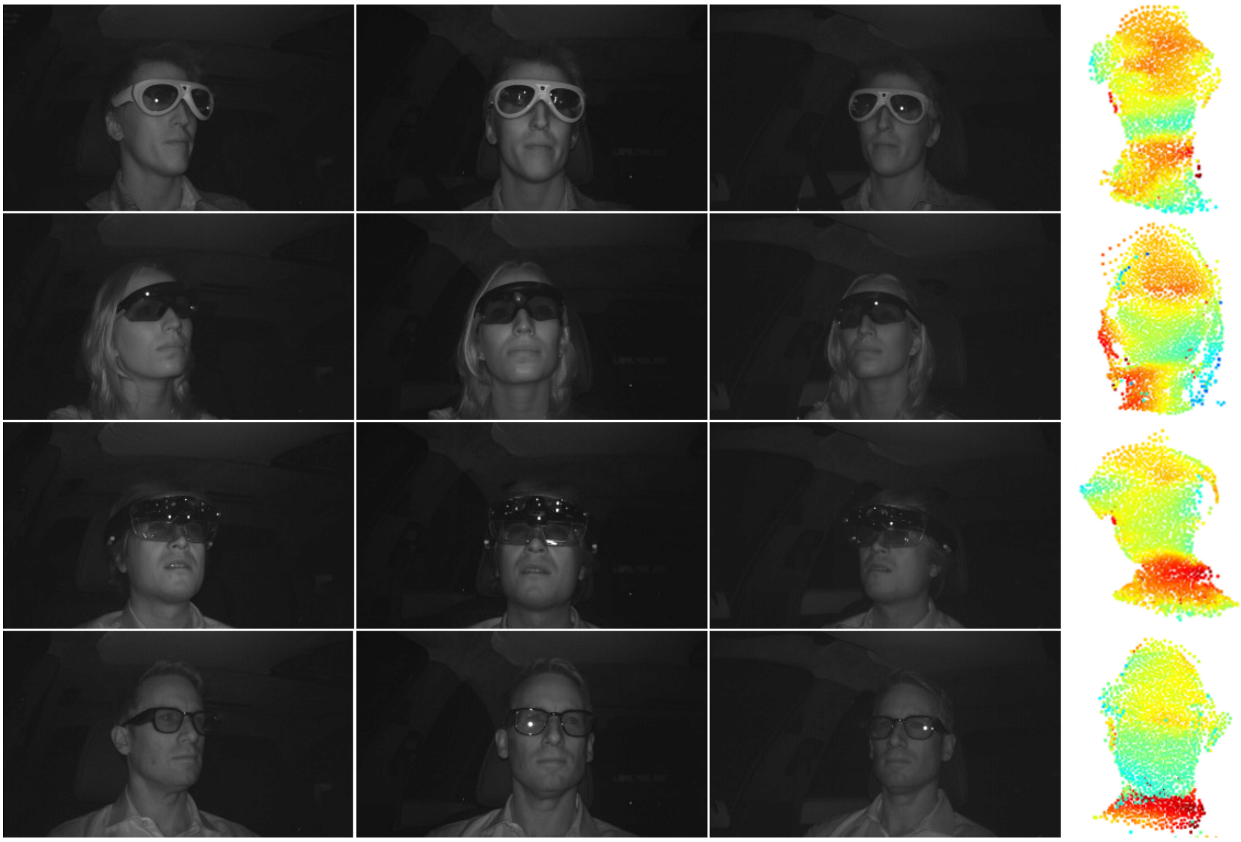


Figure 4.14: Four example triples with the estimated point cloud, predicted by the central image only. Each row visualizes an example triple for a different person, wearing different glasses and the corresponding point cloud. The lighter the blue color of 3D point, the more the relative distance of a particular 3D point.

the predicted translation is relative to the point cloud center, the absolute translation values are computed according to the predefined object box during test time.

4.3.2 Pose Representation

Based on the point clouds generated with the point cloud estimation approach, we perform 6-DoF pose estimation. Translation is represented in Euclidean space and rotation via unit quaternions for the Deep Learning-based pose estimation approaches. Quaternions are used to represent rotations due to their successful usage compared to Euler angles and rotation matrices in previous work [90, 91, 62].

4.3.3 PointsToRotation Network

A first Deep Neural Network working on the point clouds based on a PointNet++ [85] backbone and a rotation estimation module is called PointsToRotation (P2R). The input point cloud is processed by four Set Abstraction and two Feature Propagation layers. These layers where introduced by [85], specially designed to handle point set input. Stacking these layers by iteratively abstracting the input point set and then propagating the learned features, allows finding so-called seed points or critical set. A set abstraction layer is able to group an input point set based on Furthest Point Sampling and ball region clustering. Applying a PointNet afterwards helps to extract feature information. Each group found by the set abstraction layer corresponds to a local region of the input point cloud, including centroid and surrounding local feature information. Repetitive use of this type of layer helps to abstract the input point cloud and to decrease the number of points while extracting deep features for selected points. A feature propagation layer addresses the problem of obtaining deep point features for all points, based on the found features during the set abstraction step. The propagation is realized through skip connections between the original input set and the abstracted set as well as K-nearest neighbor interpolation.

Based on the backbone features, 3D rotation information is regressed from found seed points using several 1×1 convolutional layers and dense layers. Figure 4.15 visualizes the complete network architecture.

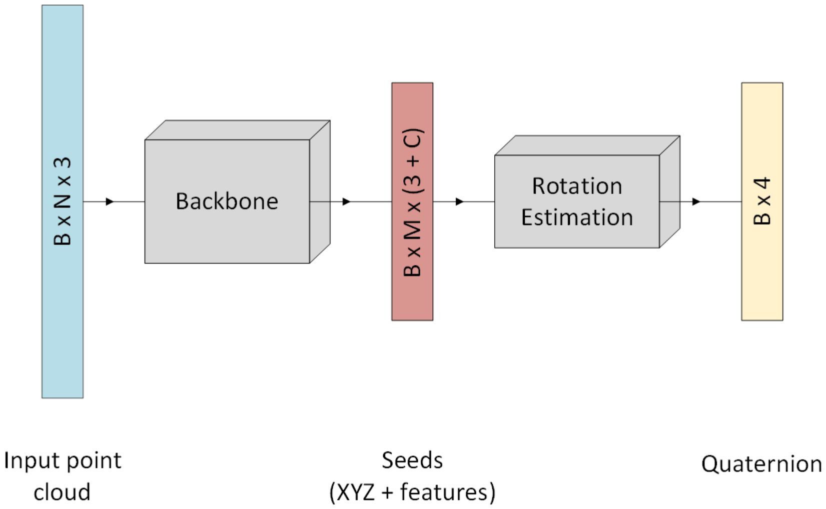


Figure 4.15: Overview of the P2R-Net architecture. N is the number of points, B refers to the batch size of the network, M is the number of seed points and C is the number of additional features found by the network.

The backbone is built of four set abstraction layers and two feature propagation layers. The progressive involvement of increasingly large regions of the point set allows for hierarchical point set feature learning [6]. Table 4.2 shows the exact parameters for the backbone layers. The ball radii for the set abstraction layers have been selected concerning the average head size of drivers and are given in meters. The ball radius parameter grows with the depth of the network from 0.02 to 1.0. The final set abstraction layer incorporates all points, such that the overall rotation of the object depends on the features of all found seed points.

In Table 4.3, the individual input and output tensor sizes per backbone layer are listed.

Layer	Num. Points	Ball radius	PointNet	Normalization
SA1	1024	0.02	[3, 64, 64, 128]	True
SA2	512	0.04	[128, 128, 128, 256]	True
SA3	256	0.08	[256, 128, 128, 256]	True
SA4	128	1.0	[256, 128, 128, 256]	True
FP1	-	-	[512, 256, 256]	-
FP2	-	-	[512, 256, 256]	-

Table 4.2: Parameters for backbone layers. For each layer, the number of points, the ball radius, the number of nodes per layers of the PointNet and normalization information is shown.

Layer	Input XYZ	Input Features	Output XYZ	Output Features
SA1	(B, 2000, 3)	None	(B, 1024, 3)	(B, 128, 1024)
SA2	SA1 out XYZ	SA1 out Features	(B, 512, 3)	(B, 256, 512)
SA3	SA2 out XYZ	SA2 out Features	(B, 256, 3)	(B, 256, 256)
SA4	SA3 out XYZ	SA3 out Features	(B, 128, 3)	(B, 256, 128)
FP1	SA3 out XYZ SA4 out XYZ	SA3 out Features SA4 out Features	(B, 256, 3)	(B, 256, 256)
FP2	SA2 out XYZ SA3 out XYZ	SA2 out Features FP1 out Features	(B, 512, 3)	(B, 256, 512)

Table 4.3: Overview of input and output for backbone layers.

The backbone used for P2R subsamples 512 seed points with a 256-dimensional feature vector from 2000 input points with original feature dimension of 3. Thus, the proposed approach keeps 25.6% of the input points. The features learned by the backbone are then fed into a rotation estimation module, which consists of three 1×1 convolutional layers with batch normalization and ReLU activation function. While the first two layers apply 256 filters each, the last layer works with 4 filters. The output is flattened and processed by two dense layers with 512 and 4 output nodes, respectively. The result is a four-dimensional vector representing a quaternion. The rotation loss $\mathcal{L}_{rot}$ normalizes the predicted quaternion and compares it to the ground truth quaternion using L_2 loss. Finally, we normalize the loss by the batch size.

4.3.4 PointsToPose Network

Building upon P2R, we introduce another Deep Neural Network, which expands the proposed network with a voting module and a translation estimation. We name the approach PointsToPose (P2P). The pipeline is inspired by VoteNet [85], where the authors argue in favor of using 3D

Hough voting to predict the centroid of 3D bounding boxes, as this point is not part of the surface, potentially acquired by a depth sensor. This results in more accurate bounding box predictions for the scene's objects than direct regression of the bounding box positions.

Compared to VoteNet [85], the input point cloud involves only ten percent of the number of points. VoteNet extracts 1024 seed points from the input point cloud with 20,000 points. A 256-dimensional feature vector characterizes each seed. The input point cloud amount is identical to the description in Section 4.3.3, starting with 2000 input points and generating 512 seed points. This approach aims to find seed points in the input point cloud and let the seeds vote for predefined keypoints K. Keypoint and voting-based object detection algorithms have proven to be successful when working with 3D point clouds. Especially the definition of keypoints benefits pose estimation when dealing with occlusions and truncation [48]. The pose estimation is assumed to be more efficient when predicted from keypoints instead of seeds. While the seed selection is heavily dependent on the input data, the definition of keypoints is independent of the input. However, it results in the same point distribution for every possible pose of the object. The keypoints are defined by subsampling a combined 3D model of a human head with glasses with Furthest Point Sampling. For this purpose, we fuse a 3D model of a synthetic male human head with a 3D model of glasses. These keypoints are the ground truth annotations for the votes. Figure 4.16 shows the network pipeline for this approach.

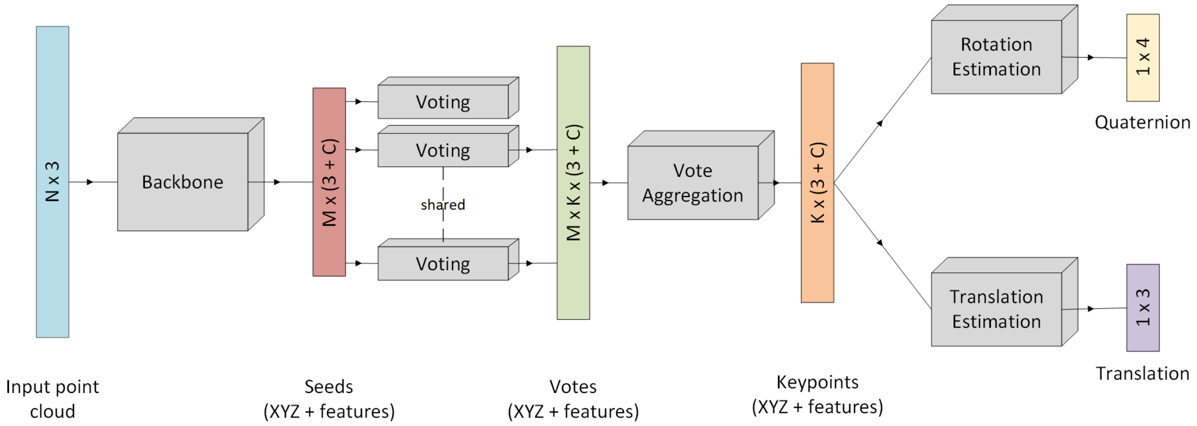


Figure 4.16: Overview of the P2P-Net architecture. N is the number of points, M refers to the number of seeds found by the backbone and K is the number of predefined keypoints.

At first, we select seed points from the input point cloud. In the next step, each seed point votes for each of the predefined keypoints. After aggregating the votes, rotation and translation information are regressed.

The loss of this network architecture is based on three different loss terms: rotation loss $\mathcal{L}_{rot}$, translation loss $\mathcal{L}_{trans}$ and vote loss $\mathcal{L}_{vote}$. The rotation loss is computed in the same manner as for P2R. The translation loss refers to the sum of the predicted relative translation and the translation of the input point cloud. The L_2 loss is used to compute the difference from ground truth translation. The vote loss supervises keypoint predictions (votes) with the ground truth position of this keypoint based on the L_2 loss. The final loss function is comprised according to equation 4.8:

$$\mathcal{L} = a * \mathcal{L}_{rot} + b * \mathcal{L}_{trans} + c * \mathcal{L}_{vote} \tag{4.8}$$

a, b and c is for the weighting of the loss terms, where we set $a = 0.2$, $b = 0.2$ and $c = 0.6$. The higher weighting of the vote loss ensures a valid base for the rotation and translation estimation. The weightings are derived through experiments. Both networks apply an Adam Optimizer with a learning rate of $\alpha = 0.001$. We train them on Nvidia Geforce GTX 1080 Ti GPUs for 30 epochs each.

4.3.5 Benchmarked State-of-the-Art Point Cloud-based Method

We additionally select a point cloud-based object pose estimation approach by Gao et al. [92] named CloudPose for benchmarking. In the original work, the authors build upon semantic segmentation and RGB-D data to derive point clouds as an intermediate representation. Then, they perform 6-DoF pose estimation based on the point clouds. A network named ”BaseNet” is being used for translation and orientation estimation separately. The network is based on PointNet [5]. For rotation estimation, they utilize the axis-angle representation and deploy the geodesic loss function. During evaluation on the defined metrics, we convert the axis-angle predictions to Euler angles.

We benchmark this method based on the generated point clouds and the ground truth pose labels. The training, validation and test split are kept identical to that used to train the P2R and P2P methods. The parameters are kept as in the original implementation for comparability.

4.4 Neural Network Ensemble Methods

We developed several Neural Network ensemble methods to analyze potential gains in performance in pose estimation when multiple networks are combined. They aim at combining various branches of AR glasses pose estimation algorithms. There are several ways to link Neural Networks. First, the outputs of the neural networks can be combined directly. We also refer to these non-learning-based approaches as ”shallow fusion”. Second, Neural Networks can be interconnected by sharing certain layers and conjointly training them to enhance the estimation. We call them ”deep fusion” methods. Thus, we first introduce multiple ways of shallow fusion, followed by learning-based deep fusion methods.

4.4.1 Shallow Fusion Methods

We developed three shallow fusion variants for later comparison to the learning-based deep fusion method: averaging the output, a weighted average, and a select best method.

Average

A simple average of predictions is the most widespread fusion operation for Neural Networks. It is important to discuss whether the model outputs are suitable for averaging. In our case, we have 7-dimensional head or HMD poses we want to average. Three values encode the position, and the remaining four dimensions parametrize the quaternion, representing the orientation. As the translation is linear, the mean is applicable. The mean of two position predictions is precisely in their middle. However, quaternions behave non-linearly, and therefore they usually cannot be averaged. Nevertheless, we have unique constraints similar to the loss function in Subsection 4.1.2, which computes the Euclidean distance between two quaternions. Mainly, the quaternions to be averaged predict similar rotations. All our models were trained with the same data and ground truth labels adapted to have a positive w component. Hence, we do not encounter the case where one network could predict a quaternion q and its negation simultaneously, why we average the quaternions identically to the position.

Weighted Average

A more sophisticated way of averaging is taking available knowledge about the values into account. Standard averaging is very susceptible to skewing the results whenever outliers are included in the data. This is a problem when dealing with a continuous prediction such as a 6-DoF pose. The error of every model will impact the accuracy of the ensemble. To remedy this, the a-priori knowledge about the expected performance of a model can be exploited to weigh its contribution to the ensemble. The main goal of ensembles is to utilize the individual models complementing strengths and reduce their weaknesses. Consequently, individual models need to be weighted accordingly. Unfortunately, DNNs are very complex, and their inner workings are often difficult to describe and analyze in any more high-level form. Since deducing a weight distribution from the inner mechanics is not possible, we focus on its output.

We partition their range and compute a weight for each subset individually. This partition should be done according to a parameter along which the accuracy changes. The error can change drastically when we move towards the edge areas of the pose. For the generation of the weights, we decided to use Euler angles as a presentation format, as the axes are separated. This allows us to treat each of the three rotational dimensions on their own without affecting another axis. For example, if a specific model predicts the yaw accurately, but another model has improved performance on the roll, we can apply different weights for each of these dimensions. In addition, HMDs often occlude parts of themselves when looking into a camera, such as the sides and the top. Some models might deduce the orientation from frontal features and their spatial relationship. Others might focus more on features that are only visible in certain poses. Hence, different models may have slightly varying characteristics for different rotation angles. To combine the predictions of our models differently per dimension, we perform the weights generation separately per dimension. We compute the mean error per "bucket" with the same parameters and gather 100 mean error values per dimension. For the fusion, we need to weigh predictions with a lower expected error higher. Therefore, we take the inverse $1/e_{m,d,b}$ for each error $e_{m,d,b}$ of model m's dimension d and bucket b. After using the prediction of every model

m_i and finding the correct bucket b for each dimension to look up all the relevant weights e_{m_i}, d, we get one $m \times d$ weight matrix. The models' weights do not sum up to 1 yet. Subsequently, we normalize them along the d-axis such that for each dimension, the weights for all models sum up to 1. Lastly, we perform an element-wise multiplication of the weights with the stacked predictions. Thus, each element is weighed and sum up the result along the m-axis to obtain the final d-dimensional pose. For the weight lookup process, the algorithm finds the bucket in which the current prediction of the current pose falls. This can be done by saving the bucket size s in millimeters or degrees and the lowest interval limit l. Then the number of the relevant bucket i can be computed directly as $i = l + \hat{y}_d / s$ and the i-th error value is used to weight its corresponding prediction.
We cannot use information from the test set to improve our model. Hence, we compute the weights with the training set, providing more values from which to compute the weights.

Select Best

Average-based fusion approaches have the inherent disadvantage of allowing comparatively worse decisions by one or multiple classifiers to impair the joint prediction of the ensemble. Although adding weights to the average can reduce the impact, it cannot completely solve it. It is solvable by choosing the best model according to an heuristic and utilize only its prediction. Our select best fusion uses the same areas for which the weights of the weighted average are computed. Instead of weighing decisions, it uses the weights as a metric to determine which individual model is the most suited for the current poses area. As we already hypothesized for the weighted average, different models might perform better or worse for a single dimension. Hence, we appoint one expert for each dimension of each input. For each model, we look up the mean error for every dimension of its prediction from the same data we already use as weights for the weighted average. The network with the lowest error passes its prediction onto the output, and all others are ignored. It is important to note that the networks appointed for an individual axis are not necessarily different. In the extreme case, one model is better than all other models on all axes and thus always chosen as the expert.

Compared to the weighted average approach, there is only one step that needs to be added. Instead of multiplying the pose estimations of all models with the weight vector directly, we first set its maximum entry for each dimension to one and all others to zero.

4.4.2 Learning-based Deep Fusion

While all previous fusions follow simple metrics to adjust the weights, there must be many more relations between poses from individual models. They can be exploited for combining the models into a better or more robust pose estimation.

Thus, we train a neural network to fuse various networks to mimic this more complex interconnection. We test two types of inputs for the learning-based method: we either utilize the pose predictions from each model only or include activations of one previous layer per network.

Fusion Network Derivation

Since our fusion gets small input data that does not have any inherently temporal or spatial relation that would justify using RNNs or CNNs, we deploy a fully connected network. We choose different numbers of layers and neurons inspired by the typical last few fully connected layers in various larger CNN-based networks. They typically get a relatively large vector of activations from the convolutional part of the model and gradually decrease its dimension until it reaches the desired output size. We use ReLU as the activation function for the layers, letting all values below zero to zero and lets other input pass through. As an alternative to directly predicting the pose, we added an option with which the model predicts a correction value that is then added to the prediction of the best model of the ensemble. This change is similar to and inspired by the residual connection in the ResNet architecture. We compared two models which are the same (input layer with 14 neurons for poses from 2 models, 2 hidden layers with 32 and 16 neurons, no dropout) except that one computes the pose and the other the pose offset. Both models use pose predictions from the SeqCapsPose and SepConv-LSTM model as input. Figure 4.17 depicts the training loss for both. It demonstrates that this small change in the network can help to converge faster and reach superior results.

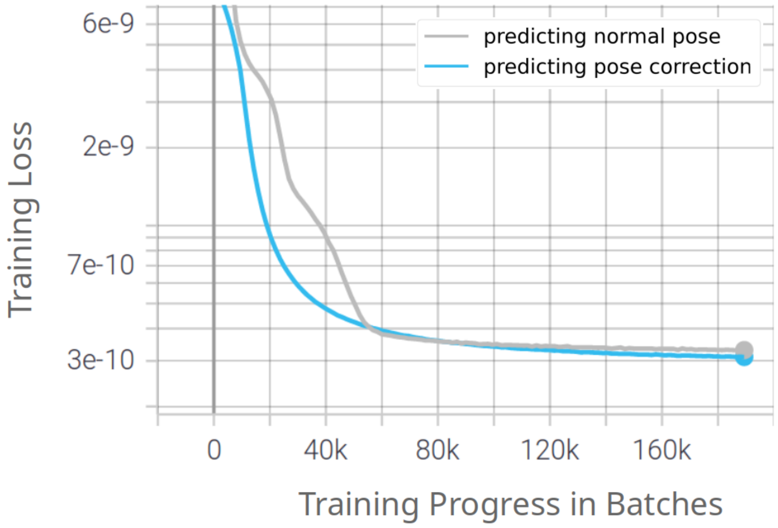


Figure 4.17: The training loss of two networks combining pose predictions from the SeqCapsPose and SepConv-LSTM models. One model computes the combined pose directly while the other outputs an offset that is then added to the SeqCapsPose pose.

The networks were developed in PyTorch Lightning [93], which offers a high-level interface to PyTorch but structures the deep learning code into more encapsulated pieces. The actual model is implemented similar to regular PyTorch, but Lightning automates parts of the training loop to avoid boilerplate code and the potential errors that come with it.

We chose the best performing four networks of different neural network categories GlassPoseRN, SepConv-LSTM, SeqCapsPose, and P2P for the first learning-based ensembles. In theory, more available information might improve the learning. Thus, we first trained our fusion

neural network with all the available data. We concatenated the final poses and the intermediate model activation from each model into a vector.
Training with this constellation resulted in a very irregular loss. Figure 4.18 presents the training loss from multiple trials with the same parameters.

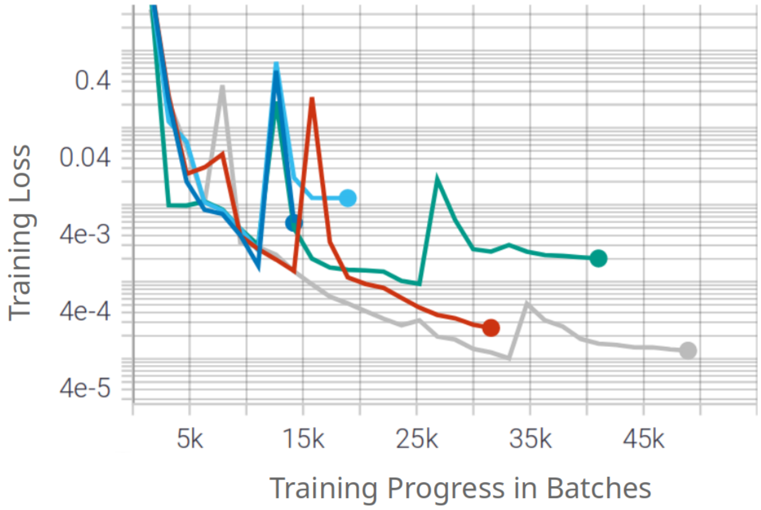


Figure 4.18: The training loss of the first trainings that take features and pose estimates from the four base models as input. The trainings are very irregular and have ”spikes”.

All loss curves have one or two ”spikes” where the loss increases drastically, sometimes reaching much lower values than other cases. This indicates that the problem in this form is possibly too complex for our type of network. With the added activations in addition to the poses, the models’ task is essentially to learn the mappings from the last few layers of four individual networks simultaneously. In order to verify that this behavior is not caused by any bugs in the framework or our implementation, we train another network that receives only the pose from our best individual model SeqCapsPose as input. We adapt its size to 2 hidden layers with 32 and then 16 neurons to reflect the smaller and less complex input. The loss of this new model converges to much lower values than before, learning to output poses that are very close to its input. From this, we conclude that the models are working well in general but cannot cope with very complex input. This problem can be solved by making the fusion network more complex. However, we assume that the input data from the considerably worse GlassPoseRN and P2P networks are not as beneficial to the result, which was already the case in our non-learning-based fusions. In addition, including the intermediate activations from the individual models of the ensemble hinder the training. Thus, we chose to apply the same kind of fusion network but feeding it with less information.

In 7.5.2, we show that selecting the best-performing models improved the performance of the shallow fusion methods. Subsequently, we only use the best two networks for the deep fusion. Hence, we generated training data from the actual pose predictions of the two best individual models and trained our final fusion network. The next training exhibited a normal progression of the loss without irregularities as before.

With the help of the Optuna-tool [94], which can perform automated hyperparameter tuning when some ranges are set beforehand, we derive our final model. We continued using the most promising combination of the pose estimators SeqCapsPose and SepConv-LSTM models as our training data. Finally, we derive the architecture as depicted in Figure 4.19.

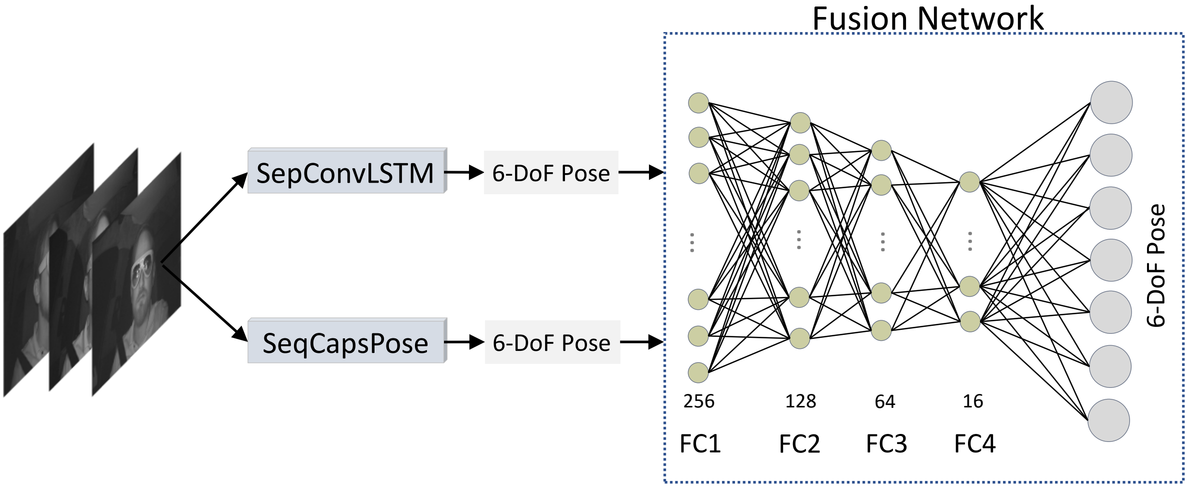


Figure 4.19: The architecture of our final deep fusion network.

We eventually deduced four fully connected layers with sizes 256, 128, 64, and 16 to work well. The final model does not include dropout and computes the pose correction instead of the pose.

Training

The Adam optimizer and a batch size of 64 was used for all trainings. The loss function was identical to the function previously presented for image-based methods. All trainings were executed on one or two Nvidia Geforce GTX 2080 Ti GPUs. We used early stopping with a patience of five epochs on the validation loss, which ended the training either when the maximum number of 120 epochs was reached or when the validation loss did not decrease further for five consecutive epochs. The final network was trained with an initial learning rate of 0.0049.

4.5 Summary

In this chapter, we introduced various ways of HMD pose estimation based on infrared images. They range from image-based feed-forward pose estimation algorithms to image-based recurrent pose trackers to point cloud-based pose estimation algorithms. Per category, we proposed various Deep Neural Networks, each utilizing different ideas of feature extraction and network architecture. Table 4.4 shows an overview of all presented algorithms. Three image-based feed-forward networks were developed and implemented. We reimplemented a state-of-the-art pose estimation method on the same data for further comparison.

Category	Approach	Input Type	Ours/ SOTA	# Images	Residual Network	Capsules	Non-Local Block	Separable Conv.
Image-based Feed-Forward Pose Estimation	GlassPose	IR Images	Ours	1-3	-	-	-	-
	GlassPoseRN	IR Images	Ours	1-3	x	-	-	-
	CapsPose	IR Images	Ours	3	-	x	-	-
	RandomizedBins [81]	IR Images	SOTA	3	-	-	-	-
Image-based Recurrent Pose Tracking	CNN-LSTM	IR Images	Ours	3	-	-	-	-
	NL-CNN-LSTM	IR Images	Ours	3	-	-	x	-
	SepConv-LSTM	IR Images	Ours	3	-	-	-	x
	NL-SepConv-LSTM	IR Images	Ours	3	-	-	x	x
	GlassPoseRN-LSTM	IR Images	Ours	3	x	-	-	-
	SeqCapsPose	IR Images	Ours	3	-	x	-	-
Point Cloud-based Feed-Forward Pose Estimation	P2R	Point Cloud	Ours	-	-	-	-	-
	P2P	Point Cloud	Ours	-	-	-	-	-
	CloudPose [92]	Point Cloud	SOTA	-	-	-	-	-
Fusion Methods	Average	Neural Net	Ours	-	-	-	-	
	Weighted Average	Neural Net	Ours	-	-	-	-	-
	Select Best	Neural Net	Ours	-	-	-	-	-
	Learning Fusion	Neural Net	Ours	-	-	-	-	-

Table 4.4: An overview of all presented and trained algorithms. ”SOTA” stands for state-of-the-art, ”Separable Conv.” for Separable Convolutions.

In the image-based recurrent pose tracking category, we presented three different ways of pose regression based on Recurrent Neural Networks. CNNs with LSTM additions, Non-Local Blocks, or Capsule Networks with additional LSTM layers were introduced.

Finally, the derivation of point cloud based on a semi-supervised network with consecutive pose estimation networks was shown. One algorithm regresses the orientation based on a CNN, while another network introduces a voting-based mechanism for 6-DoF pose regression. A state-of-the-art, PointNet [5] based network is used for comparison.

We then introduced Ensemble methods for further analysis of potential performance gains when different networks are combined. There are non-learning-based methods, mainly combining the outputs of the individual networks. Learning-based ensemble methods try to interconnect the networks further by sharing layers, which are trained conjointly.

Before evaluating the AR glasses pose, we introduce head pose methods in the next chapter. Some methods introduced for AR glasses pose estimation are directly used for head pose estimation.

5 Deep Learning-based Head Pose Estimation

Supplementary to 6-DoF AR glasses pose estimation, the head pose can potentially be used in cases where the HMD pose is inaccurate. For this purpose, we additionally analyze head pose estimation.

First, we propose various methods for head orientation estimation only. They served as the foundation for the AR glasses pose estimation algorithms, as we included findings from the networks into our HMD pose estimators. The head pose methods are trained on the AutoPOSE dataset.

Second, the algorithms used for AR glasses pose are adjusted and retrained for 6-DoF head pose estimation. We compare their performance to HMD pose estimation in Chapter 7. Finally, the chapter presents a method to fuse the head and HMD pose.

5.1 In-Car Head Pose Estimation on the AutoPOSE Dataset

For a first head pose estimation analysis, we introduce different Deep Learning algorithms trained on the AutoPOSE dataset [95]. The dataset AutoPOSE provides around 1.1M 752 ×480 pixel IR images of 21 subjects and was recorded in a car simulator. The dataset consists of highly accurate head pose annotations synchronized with the IR images.

We use different networks on the IR data to perform and evaluate head pose estimation. We conduct preprocessing on the raw images, where we clean the images first based on head visibility. Afterwards, we generate head-cropped images.

5.1.1 Dataset Preparation

In a first preparation step, we sort out frames, where we keep the frames with rotations higher than 120 degrees for training to increase robustness but eliminate them from the validation and test set. In addition, we equalize and normalize the images.

Borghi et al. use the output of a different neural network to compute the 2D head position, which they then use for cropping the image [71, 96]. Similarly, any open source library such as [97] can be used to find a head bounding box. Since the orientation is more volatile and more crucial in a driving scenario, we focus on orientation regression for a first insight into in-car IR image-based regression. We do not want to add imprecision through position estimation in this orientation evaluation. Thus we do not perform head position estimation. Instead, we obtain the head center from the ground truth data. This prevents having additional error in the pose estimation part introduced through another position estimation method. Subsequently, we determine the head center in image coordinates (x_H, y_H). The head bounding box is deduced from the acquired head center, defined by the width w_H and the height h_H, used to crop the frames. The horizontal and vertical focal lengths of the acquisition device, distance D between the head center and the acquisition device, and R_x and R_y, which are the average width and height of a face, help deduce a dynamic size bounding box. The head width R_x and height R_y in 3D is defined uniformly as 32 cm, so the head is equal in size inside the cropped images. Additionally, we discard the cropped image if more than a third of the head is not visible in the frame. We generate two options to evaluate on different resolution levels, one being 64×64, the other 128×128 pixels. We train and evaluate a variety of networks on the generated images for 3D head pose regression.

5.1.2 Neural Networks for Head Orientation Estimation

For later evaluation, we benchmark existing methods and develop our own methods for head orientation regression on the AutoPOSE dataset. The first network is the POSEidon model [71, 96], which we train and test on 64×64 pixel images. As that model does include a sufficient amount of layers to handle 128×128 pixel images, we develop our own network called Head Orientation Network (HON), which has a similar architecture. Furthermore, we benchmark the HPN model, which was developed for the DriveAHead dataset [70], on both resolution levels. Finally, we implement a Deep Neural Network based on ResNet [3] and Hourglass [98] network, suitable for 64×64 and 128×128 cropped pixel images of the AutoPOSE dataset.

POSEidon model

We used the POSEidon-CNN on the IR data to perform head pose estimation [71, 96]. The framework relied on depth data and did not perform Head Pose Estimation on IR images. The head pose estimation part in the framework is based on three different branches, which consider depth maps, grayscale images generated from depth maps, and motion images. All branches were trained with the same CNN architecture. The output of the three branches is fused in the end. We obtained the CNN, which each branch in the framework used separately (Figure 5.1). The network contains five convolutional layers with three consecutive fully connected layers for orientation estimation. The first two layers have a filter size of 5×5, the third layer a size of 4×4, and the last two convolutional layers a filter size of 3×3. The first three convolutional layers are followed by max pooling layers to reduce the layer sizes. The model exploits Dropout as regularization ($\sigma = 0.5$) at the two fully connected layers. The Deep Neural Network was trained on the cropped images of the dataset with a resolution of 64×64.

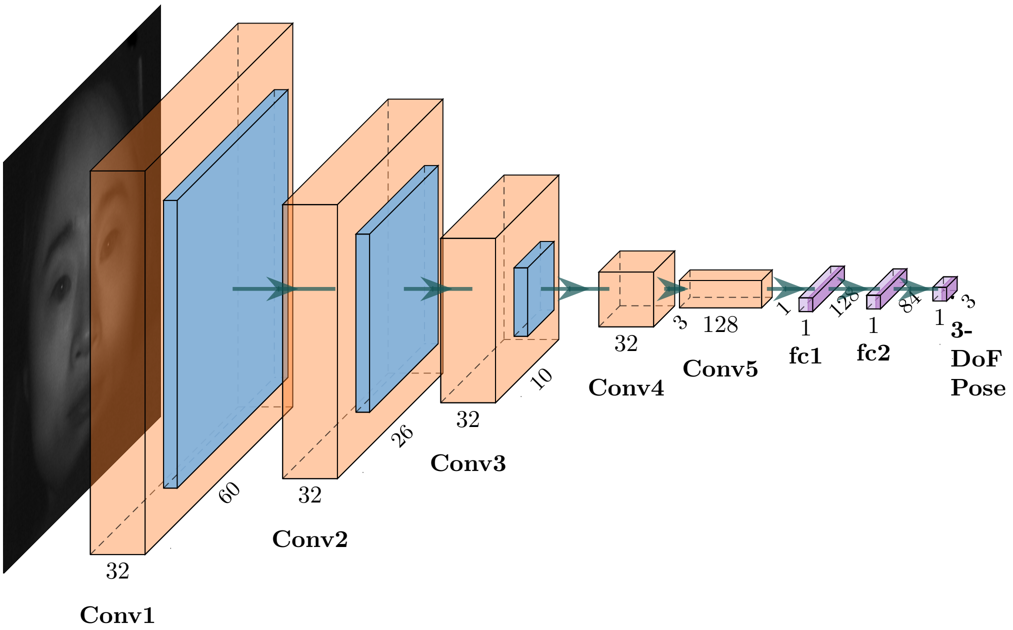


Figure 5.1: The VGG-based CNN used in the POSEidon-framework [71, 96]. The orange layers are convolutional layers, the blue layers max pooling layers, and the purple layers fully connected layers.

Head Orientation Network - HON

We design our own, efficient network named "Head Orientation Network" (HON) inspired by VGG [38] and the model of the POSEidon-framework [96] (Figure 5.2). We use this model only on 128×128 pixel images, as the VGG-based POSEidon network is of similar shape and is used for 64×64 pixel cropped images. The network includes 6 convolutional layers with three consecutive fully connected layers for orientation estimation. The first three layers have a filter size of 5×5, while the last three convolutional layers have a size of 3×3. The first three convolutional layers are followed by max pooling layers to reduce the layers sizes. We train HON with an Adam optimizer with the initial learning rate $\alpha = 0.0001$. We exploit Dropout as regularization ($\sigma = 0.5$) at the two fully connected layers.

HPN model (DriveAHead)

We additionally consider one of the most recent, learning-based in-car head pose estimation algorithms on IR data: The HPN model [70, 91], originally created as a baseline estimator for the DriveAHead dataset. We reimplement and train the model from scratch on the AutoPOSE dataset with the same initial learning rate $\alpha = 0.001$ with the Adam optimizer [99]. We only change the output layer to regress Euler angles to match all other models in this subsection, which regress Euler angles. We perform no further changes to the model. Finally, we develop

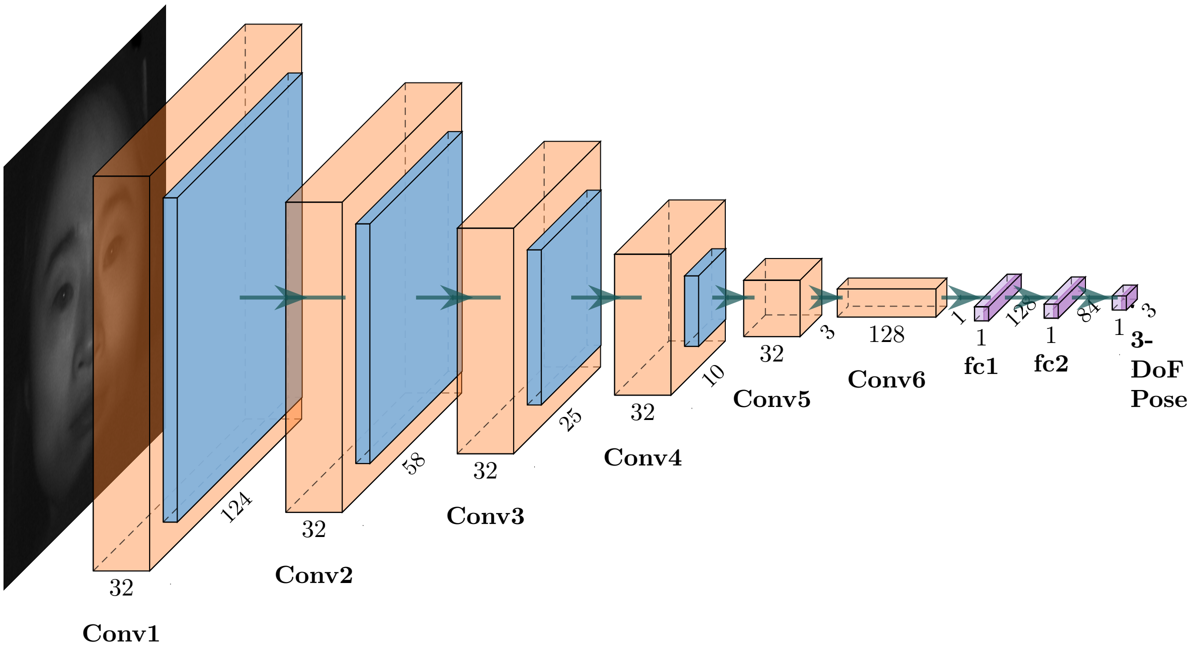


Figure 5.2: The Head Orientation Network for 128 pixel images. The orange layers are convolutional layers, the blue layers max pooling layers, and the purple layers fully connected layers.

one more novel approach for the head pose estimation task.

ResNet-based model

As an alternative to the models mentioned earlier, we developed a model based on ResNet [3] and Hourglass [98] architectures. The model maintains information and feature within building blocks of ResNet with skip connection as described in [3]. In addition, lower-level features from the head of earlier layers are being connected with later layers working on higher-level head features with hourglass-like skip connections [98]. Thus, coarse and fine-grained features of the head are being utilized for a head pose regression. Figure 5.3 shows an overview of the architecture. We refer to the model as ”ResNetHG” in general and ”ResNetHG18” for the ResNet-18 variant in the following. We realize the additional two skip connections by first applying a Convolution with a stride of 8 or 2, respectively. Then, the output of the source block is being added to the output of the destination block, additionally applying the ReLu-Activation.

We trained and tested the described models on cropped images from the AutoPOSE dataset on two different scaling levels.

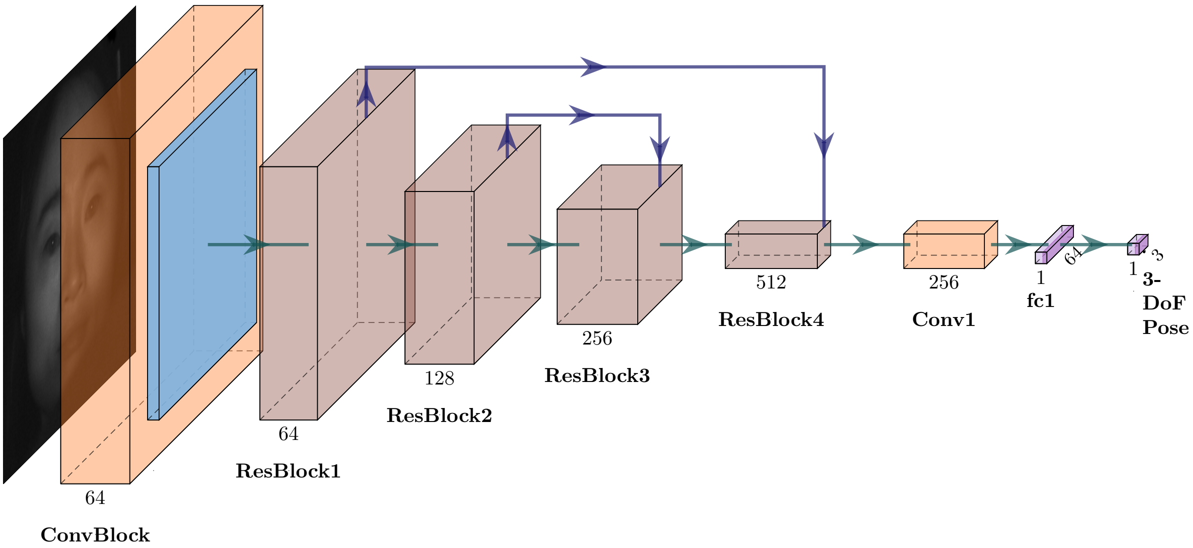


Figure 5.3: The ResNetHG18 architecture. The ”ConvBlock” and the ”ResBlock”s are as described in [3]. We added connections from ”ResBlock1” to ”ResBlock4” and ”ResBlock2” to ”ResBlock3”. The model is suitable for both cropped image sizes.

Training Details

All models were trained with the same loss function:

$$loss := \sum_{i=1}^{3} ||w_i \cdot (y_i - f(x_i))||_2 \tag{5.1}$$

where $w_i \in [0.45, 0.35, 0.2]$. For training, the loss function was defined as presented by Borghi et al. [71, 96] to focus on the yaw, which is predominant in the automotive context. Thus, we used a weighted L_2 loss between label and prediction, where we weighed the difference between them on the yaw with $w_i = 0.45$, pitch with $w_i = 0.35$ and roll with $w_i = 0.2$. Our labels range from -180 degree to 180 degree. Furthermore, 19 of the 21 sequences were taken of the subjects for training. We use one sequence for the validation set and one for testing. Thus, the test and validation sets consist of around 50k images each, whereas the rest is used for training. The training was done in batches with a size of 128, where the batches were chosen randomly. We trained the models on Nvidia Geforce GTX 1080 Ti GPUs until convergence.

5.2 Adaptation of AR Glasses Pose Techniques for Head Pose

The approaches presented in the previous section were the foundation for the final image-based pose estimation algorithms used for HMD and head pose estimation. Thus, the methods presented in Chapter 4 were also used for head pose estimation, partially with modifications. This section outlines the specific methods and their adaptations used for head pose estimation. We add "-HMD" or "-HEAD" to the specific method to differentiate between the respective versions. The suffixes are added whenever both HMD and head pose estimation are discussed simultaneously. Subsequently, we present adaptation techniques applied to the presented AR glasses pose estimators to train them on a head pose dataset. The head pose annotations were acquired based on the HMDPose dataset. The annotation generation pipeline will be presented in Chapter 6.

5.2.1 Image-based Feed-Forward Head Pose Estimation

We first introduce the image-based feed-forward methods and their adaptations for head pose estimation. This includes the GlassPose, GlassPoseRN, and CapsPose methods. They are called "GlassPose-HMD", "GlassPose-HEAD", "GlassPoseRN-HMD", "GlassPoseRN-HEAD", "CapsPose-HMD", and "CapsPose-HEAD".

GlassPose-HMD & GlassPose-HEAD

In Subsection 4.1.1, a CNN called GlassPose for glasses pose estimation was presented. For simplicity, we call this network GlassPose-HMD, and the head-custom version GlassPose-HEAD. We modify its architecture to fit the generated head pose version of the HMDPose dataset for head pose estimation purposes. We call the adapted version GlassPose-HEAD network.

The GlassPose-HMD estimation approach consists of two decoupled sub-networks for estimating the glasses' position and orientation. The first sub-network for the translation estimation task inputs a down-sampled version of triple infrared images with a 320×188 pixel resolution. The second sub-network for orientation estimation uses an automatic cropping module [77]. The glasses are extracted from the cropped faces and are fed to the orientation sub-network. The resolution of the glasses images is 128×54 pixels. We filter out the glasses from the detected faces before feeding them to the GlassPose-HEAD network to guarantee a fair comparison between the glasses and head pose estimation. That ensures the network to extract features from the head only. The resolution of the head images fed to the orientation sub-network of GlassPose-HEAD is 128×172. The difference in the cropping process is shown in Figure 5.4. Both networks have the same architecture and differ only in the size of the input layer.

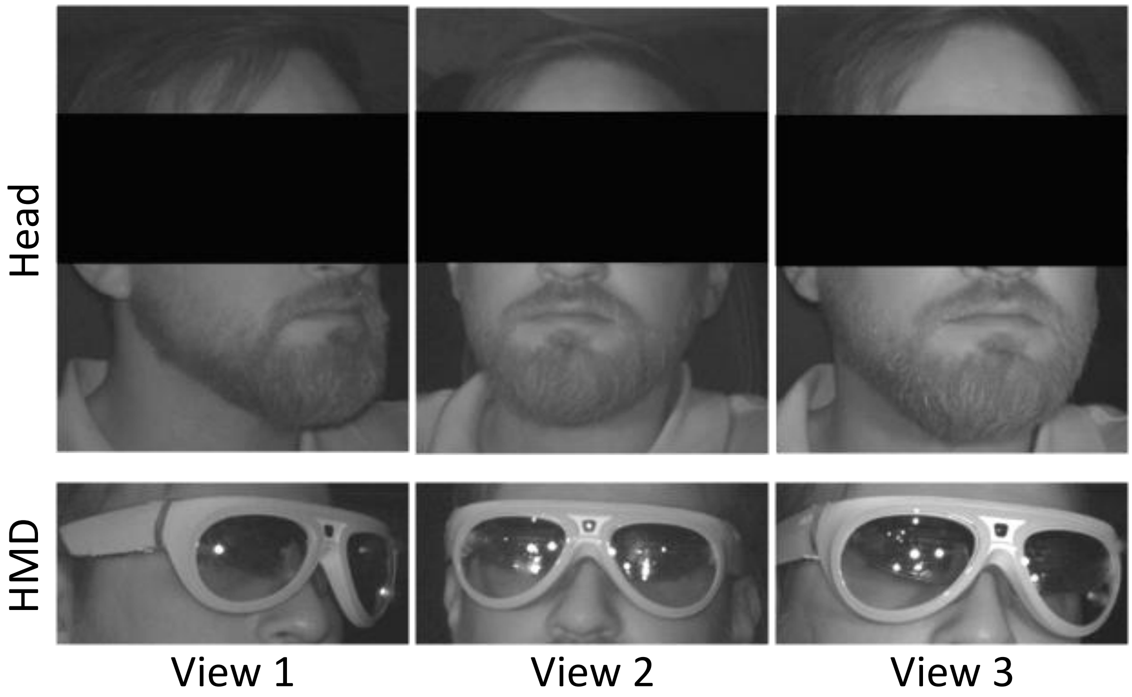

[redacted] [redacted] [redacted]

Figure 5.4: Cropping step used in GlassPose-HMD and GlassPose-HEAD models. Cropped heads with filtered glasses are fed to the orientation part of GlassPose-HEAD. GlassPose-HMD utilizes cropped glasses for the rotation estimation.

GlassPoseRN-HMD & GlassPoseRN-HEAD

The GlassPoseRN method for AR glasses pose estimation was presented in Subsection 4.1.2. When a distinction of the head pose variant is required, we call this network GlassPoseRN-HMD. We train this network for head pose estimation and call it GlassPoseRN-HEAD. GlassPoseRN-HMD networks use ResNet-18[3] as a backbone for feature extraction consisting of a convolutional block followed by four residual blocks. The backbone is followed by three fully connected layers with 256, 64 and 7 neurons, respectively. GlassPoseRN-HEAD has the same architecture.

CapsPose-HMD & CapsPose-HEAD

CapsPose is a network that utilizes Capsules for pose estimation, as introduced in Subsection 4.1.3. When a distinction of the head pose trained network and the HMD pose networks is required, we call this network CapsPose-HMD and the model trained on head pose labels CapsPose-HEAD. CapsPose-HMD has two convolutional layers to extract features from the input images. Then, Primary Capsules are built, that represent the orientation and position of the features of the HMD in a local coordinate system. By deploying dynamic routing, they are routed to their Pose Capsules, representing higher-level Capsules, which embody the pose of the glasses in its own Capsule Space. A projector module maps the poses from the Capsule parameter space to the desired 6-DoF pose space. CapsPose-HEAD is trained on the same architecture.

Training Details

The networks are trained similar to the description in Section 4.1 on Nvidia Geforce GTX 2080 Ti GPUs. GlassPose, GlassPoseRN, and CapsPose are trained for 180 epochs with a batch size of 128. We use the Adam optimizer with the learning rate $\alpha = 0.0001$. We train the networks using a weighted sum of the translation and rotation losses, which is identical to the HMD variants of the networks.

5.2.2 Image-based Recurrent Pose Tracking

Secondly, we present the image-based recurrent pose tracking methods and their adaptations for head pose estimation. This includes the SepConv-LSTM and the SeqCapsPose methods. They are called ”SepConv-LSTM-HMD”, ”SepConv-LSTM-HEAD”, ”SeqCapsPose-HMD” and ”SeqCapsPose-HEAD”.

SepConv-LSTM-HMD & SepConv-LSTM-HEAD

We choose SepConv-LSTM among the methods introduced in Section 4.2 first to adapt it for head pose tracking. We refer to SepConv-LSTM-HMD and SepConv-LSTM-HEAD as the models trained for glasses and head pose estimation, respectively.

This model is composed of two parts. The first part is a block of convolutional layers, which inputs a sequence of channel-wise stacked triple images and outputs a sequence of spatial features. This block has four convolutional layers. A max pooling layer follows each layer to reduce the dimensionality of the feature maps. Instead of using standard convolutional layers, this model utilizes separable depth-wise convolutional layers. The information across the channels is separated to decouple the 3-input images, as processing them jointly may cause ambiguities. Since the output channel of the depth-wise convolution is equal to the input channel, a point-wise convolution operation is appended to increase the number of channels yielding more feature maps. The second part is a block of two LSTM layers. The output is given to a dense layer to regress the pose.

SeqCapsPose-HMD & SeqCapsPose-HEAD

A second recurrent pose tracking technique chosen for head pose estimation is SeqCapsPose, also introduced in Section 4.2. SeqCapsPose-HMD describes the model trained on AR glasses, the SeqCapsPose-HEAD the model trained for head tracking.

This method incorporates the CapsPose network with the addition of LSTM layers. Two convolutional layers extract low-level features, followed by two Capsule layers to learn spatial information. The second Capsule layer includes a pose prediction in Capsule space. This Capsule backbone exploits the full spatial information. Instead of a non-recurrent projection from the Capsule space to the target pose space as done in CapsPose, the recurrent block is added. It learns the correlation of the Capsule codes in time. That block consists of two LSTM layers. With the help of the LSTM layers, the short and long-term dependencies are learned.

Training Details

We train the SepConv-LSTM model for 180 epochs using an Adam optimizer We on Nvidia Geforce GTX 2080 Ti GPUs with a scheduled learning rate, which is initially set to $\alpha = 0.001$ and is decreased if the validation loss has not improved during the last ten epochs. The same loss is used for SepConv-LSTM-HEAD, as already done for SepConv-LSTM-HMD.

The SeqCapsPose-HEAD model is also trained identically to SeqCapsPose-HMD on Nvidia Geforce GTX 2080 Ti GPUs. An Adam optimizer with the learning rate $\alpha = 0.0001$ is used for 180 epochs, with the same loss function.

5.2.3 Point Cloud-based Pose Estimation

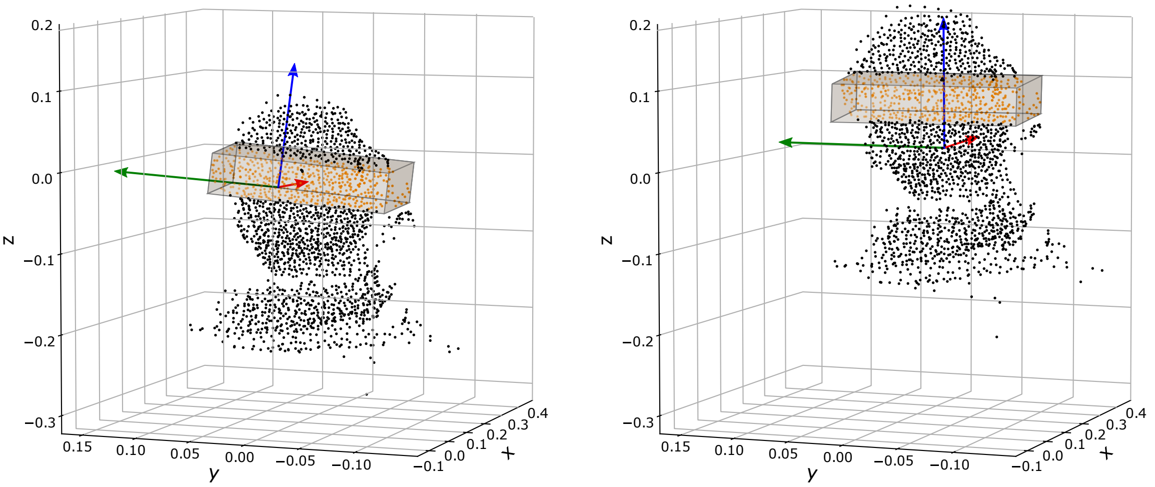
Lastly, one point cloud-based pose estimation approach is adapted for head pose estimation. We train the HMD and head pose estimator networks with the full point cloud of the head and a filtered point cloud each. Thus, we introduce ”P2P-F-HMD”, ”P2P-F-HEAD” for the networks using the full point cloud, while we name the networks with partial point clouds ”P2P-P-HMD” and ”P2P-P-HEAD”.

P2P-HMD & P2P-HEAD

We use the Point Cloud to Pose (P2P) estimation approach proposed in Section 4.3. For this method, point cloud data is generated first from the triple infrared images in a semi-supervised fashion. For comparison, we select cloud points of either the HMD or the head. We also train both networks with full point clouds. The point selection is a deterministic process that follows predefined steps. The represented target point cloud $PC_T = \{{}^1\mathbf{c}^T, \ldots, {}^N\mathbf{c}^T\}$ is the result of a Euclidean transformation $\boldsymbol{T}_S^T$ embodying the rotation $\mathbf{R}$ and the translation $\mathbf{t}$ applied on a source point cloud $PC_S = \{{}^1\mathbf{c}^S, \ldots, {}^N\mathbf{c}^S\}$, where the following holds for an arbitrary point i:

$$ {}^i\mathbf{c}^T = \mathbf{R}\,{}^i\mathbf{c}^S + \mathbf{t}\,. \tag{5.2} $$

A source point cloud PC_S represents a centered head wearing the glasses and facing the camera. Hence, the translation $\mathbf{t}$ is given by the mean of the target point cloud. In other words, it holds $\mathbf{t} = \frac{1}{N}\sum_{i=1}^{N} {}^i\mathbf{c}^T$. We get the source point cloud by subtracting the measured translation $\mathbf{t}$ from the target point cloud and multiplying the result by the inverse rotation $\mathbf{R}^T$ obtained from the HMDPose dataset. We manually divide the cloud points into two groups: points of the head and points of the glasses. The first 20% of the source point cloud starting by counting from the top are points of the head. We label the following 450 points as points of the glasses. The rest of the points are assigned to the head class, which results in 1550 head points. Then, the selected point clouds are transformed back to the target space. Consequently, we obtain two disjoint point cloud datasets PC^{HMD} and PC^{Head} corresponding to the glasses and the head, where $PC^{T} = PC^{HMD} \cup PC^{Head}$. Figure 5.5 shows the source and the target point clouds, and the labeling 3D bounding box.



(a) Target Point Cloud

(b) Source Point Cloud

Figure 5.5: Points class assignment: orange points belong to the HMD point cloud. Points in black belong to the head point cloud.

We adapt the P2P-network by changing the input shape. We call the original model P2P-F-HMD, as it uses the full point cloud for glasses pose estimation. The corresponding model for head pose estimation is called P2P-F-HEAD. We denote the models that use a subset of the cloud by P2P-P-HMD and P2P-P-HEAD for glasses and head pose estimation, respectively.

The presented P2P architecture is inspired by VoteNet [85]. A backbone sub-samples the full point cloud and learns point cloud features using the PointNet++[6]. The selected point clouds are called seed points. Each seed point provides a vote for predefined key points. The key points are defined by sub-sampling a 3D head model with glasses. A proposal module aggregates the key points to regress the pose. The model was trained by minimizing a weighted loss consisting of the translational, rotational, and vote loss, based on the ground truth positions of the key points in the 3D model.

All networks are trained for 30 epochs on Nvidia Geforce GTX 2080 Ti GPUs each using an Adam Optimizer with a learning rate of $\alpha = 0.001$ by using the same loss function as described in Subsection 4.3.4.

5.3 Fusion of HMD and Head Pose

Combining the head and HMD pose estimation potential seems theoretically promising, as both information is available in an image while driving with AR glasses. In the following, we create an ensemble of two SeqCapsPose networks, one trained on head and the other on glasses pose. The head pose and HMD pose weights are already available, why we do not need to train any base network for the fusion.

However, as we will discuss in Subsection 7.4.2, the head-trained variant performs significantly worse than the SeqCapsPose-HMD. This points towards a low probability of improving the overall result by fusing the two networks. Nevertheless, for the sake of completeness, we conduct this fusion.

In order to combine both of these models, we first have to compute the HMD pose from the head pose prediction:

$$T_{HMD,Pred} = T_{Head,Pred} + \Delta T \tag{5.3}$$

$T_{HMD,Pred}$ is the target HMD position, while $T_{Head,Pred}$ is the head pose position. ΔT depicts the transformation required to acquire $T_{HMD,Pred}$ from $T_{Head,Pred}$. The reason for the necessity to compute this is that head and HMD do not coincide. Glasses sit on the wearers nose and ears or head, which usually means they are slightly shifted forward, upward and are also rotated. The glasses in the recorded images of the HMDPose dataset are slightly shifted forwards (along the nose of the person) which can be seen when looking at the offset between the z-axes of both coordinate systems. Apart from that, the main difference is the difference in pitch, $\Delta Pitch$. The y-axes of both objects are rotated relative to each other in a way that suggests a difference in yaw between HMD and head. This offset between both coordinate systems has to be added to the head pose prediction in order to generate a HMD pose prediction from it. This can then be combined with the HMD pose predicted directly.

While AR glasses sit at around the same place on every subject's face, there are still small deviations. Firstly, depending on face shape and preference of the person wearing it, the position might be slightly different. Furthermore, the AR glasses position and rotation can shift over time. For example, it can slide downward along the nose, be intentionally repositioned by the subject or unintentionally shifted by the acceleration of the head [100]. Therefore, the assumption that the offset between the pose of the head and the glasses is constant necessarily introduces errors.

Our first approach to compute this offset takes the mean distance (in translation and orientation) between head and glasses over the complete training set:

$$\Delta T_1 = \frac{1}{n}\sum_{i=0}^{n}(T_{GT,HMD,i} - T_{Pred,Head,i}) \tag{5.4}$$

Instead of introducing another ground truth, we assume that the prediction error of the head pose is averaged out while computing the mean of the training data. Thus, we subtract the predicted head pose from each data point's ground truth HMD pose and average this difference. We transformed the orientation to Euler angles for these computations.
The first, simple approach has a conceptual flaw that can be circumvented. The head orientations of the subjects in our dataset are centered around a mean. As Table 5.1 demonstrates, the glasses are slightly shifted on the subjects' heads and mainly a bit forward towards the camera. However, this only makes sense for frontal poses. When looking, for example, 90 degrees to

		Approach 1	Approach 2
Mean Rotational Offset in Degrees	Roll	0.567	
	Pitch	10.129	
	Yaw	-0.255	
Mean Translational Offset in cm	x	-0.994	-0.674
	y	-1.869	-1.674
	z	5.869	5.699

Table 5.1: The mean offset from head to glasses, computed naively (Approach 1) and with taking the head orientation into account for the computation of the translational offset (Approach 2).

the right, the HMD would no longer be shifted mainly towards the camera but to the right, and the shifts in the other directions would also change accordingly. This scenario leads to two conclusions. Firstly, when adding the offset for the glasses to the head position, the head position should be considered. Secondly, for an optimal computation of the mean offset, the same is true. Otherwise, the non-frontal poses contribute very different offsets, which in the best case average out. Then, we do not use available information for the offset computation, which is not optimal but would be tolerable due to a large amount of available data. However, if the distribution of the orientation is not symmetrical, the result will also be skewed. This is the case in some dimensions of our data.

We address both these concerns in our second approach by computing the offset:

$$\Delta T_2 = \frac{1}{n} \sum_{i=0}^{n} (R_{Head,Pred} * (T_{GT,HMD,i} - T_{Pred,Head,i})) \tag{5.5}$$

We transform both translations $T_{GT,HMD,i}$ and $T_{Pred,Head,i}$ with the rotation matrix $R_{Head,Pred}$ that corresponds to the orientation prediction for the head pose. The rotation from the ground truth is more accurate but not usable in this case, as it is not available during inference. Instead of directly adding the offset to the prediction like previously, we have to transform the position prediction $T_{Pred,Head,i}$ by the orientation prediction $R_{Head,Pred}$, add the offset and transform the result back, undoing the rotation again. The rotational offset can be added directly without the transformations. We want the axes of the coordinate frame, that we compute the difference in, to stay consistent. It does not necessarily have to be aligned with the orientation of the head or glasses. We use the head pose as the HMD pose is not available during inference. Table 5.1 shows our results for the head to glasses offset.

The HMD is mainly tilted downward by around 10 degrees and sits 5-6 cm in front and 1-2 cm below the center of the head. The translational offset differs by 2-3mm for both of our approaches. In 7.5.2, we evaluate how this translates to the actual HMD pose prediction.

5.4 Summary

This chapter first presented several Deep Neural Networks for Head Pose estimation, trained on a head pose dataset called AutoPOSE [95]. The approaches are image-based CNNs and estimate the orientation of the head based on either 64 × 64 or 128 × 128. The images are cropped to the head area and then rescaled to two resolution levels. Among the methods is the POSEidon [71, 96] model, usable for 64 × 64 pixel images. We develop our own efficient CNN named HON with an additional convolutional layer to handle 128 × 128 image inputs. Another method is the HPN network introduced for the DriveAHead dataset [70], which can be used for both resolution levels. We further present ResNetHG, which works on both resolution levels and is a ResNet [3] and Hourglass [98] -based architecture. The presented methods were a foundation for the image-based AR glasses pose estimators introduced in Section 4.1.

In the next step, the AR glasses models were adapted for head pose estimation. To enable a distinction, we add the suffix ”-HMD” for models trained on AR glasses labels, while we add ”-HEAD” to the networks trained on head labels. Approaches ranged from image-based feed-forward pose estimators to image-based recurrent pose trackers to point cloud-based pose estimators. While some methods did not require further adjustment other than training on head pose labels, we retrained some networks on different input. Namely, the GlassPose and P2P pose estimators were adjusted for head pose estimation. GlassPose required a different extract of the image for its orientation estimation part. In return, P2P was trained with the full point cloud for head and HMD pose estimation but also trained on partial point clouds, where the part depicting the head or the glasses was cut out.

Finally, we presented fusion methods for head and HMD pose estimation to potentially enhance overall tracking accuracy. This is done by generating the HMD pose from the head pose based on the SeqCapsPose model and later combining the pose predictions. Table 5.2 shows an overview of all presented algorithms.

Category	Approach	Input Type	Dataset	Ours/ SOTA	# Images	Residual Network	Capsules	Separable Conv.
Image-based Feed-Forward Pose Estimation	POSEidon [71]	IR Images	AutoPOSE	SOTA	1	-	-	-
	HON	IR Images	AutoPOSE	Ours	1	-	-	-
	ResNetHG	IR Images	AutoPOSE	Ours	1	x	-	-
	HPN [70, 91]	IR Images	AutoPOSE	SOTA	1	-	-	-
	GlassPose	IR Images	HMDPose	Ours	3	-	-	-
	GlassPoseRN	IR Images	HMDPose	Ours	3	x	-	-
	CapsPose	IR Images	HMDPose	Ours	3	-	x	-
Image-based Recurrent Pose Tracking	SepConv-LSTM	IR Images	HMDPose	Ours	3	-	-	x
	SeqCapsPose	IR Images	HMDPose	Ours	3	-	x	-
Point Cloud-based Feed-Forward Pose Estimation	P2P	Point Cloud	HMDPose	Ours	-	-	-	-

Table 5.2: An overview of all presented and trained head pose algorithms. ”SOTA” stands for state-of-the-art, ”Separable Conv.” for Separable Convolutions.

In the next chapter, the HMDPose dataset containing infrared image triples with HMD pose annotations will be introduced first. The dataset constitutes the foundation for the network training. The chapter additionally contains our head pose label generation pipeline. The head pose labels were used to train the networks presented in Section 5.2.

6 Large-Scale 6-DoF HMD and Head Pose Dataset

The introduced pose estimation algorithms are all Deep Learning-based methods requiring vast amounts of data for proper training. For this purpose, it is essential to gather a large-scale dataset, which enables training of deep AR glasses and Head pose estimation neural networks.

This chapter will present the HMD dataset in Section 6.1 first, which contains infrared images from three different angles of 14 people wearing four AR glasses models. The dataset consists of more than 3,000,000 infrared images with 6-DoF AR glasses pose annotations. The corresponding hardware setup, the calibration and synchronization steps, the acquisition, and postprocessing of the AR glasses dataset are part of this section.

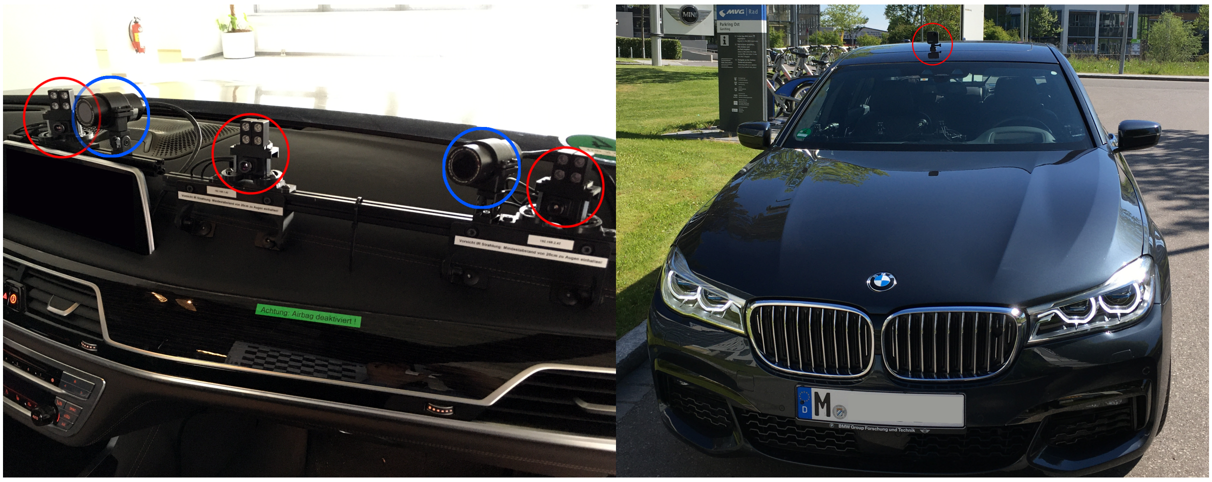
In the second part of the chapter, the associated head pose data will be introduced, generated from the same infrared images of the dataset. We detail our pipeline by demonstrating the facial landmark extraction and reconstruction, as well as the post processing steps. We finally compare the HMD and head annotations.

6.1 HMDPose - an AR Glasses Dataset

For the purpose of training the AR glasses pose Deep Neural Networks, we recorded a large-scale dataset named ”HMDPose”. It contains triple IR images of participants wearing smart glasses, appended with the associated 6-DoF AR glasses pose. The HMDPose dataset is publicly available at ags.cs.uni-kl.de/datasets/hmdpose/. The recording of the dataset required a sophisticated setup.

6.1.1 Hardware Setup

We deployed a marker-based solution for acquiring the 6-DoF-Pose as ground truth and captured the images with three infrared cameras in a car (Figure 6.1). Three IR-cameras record images with the resolution of 1280×752 pixel and 60FPS each. Furthermore, the ART-branded marker-based cameras track the markers on the glasses with high accuracy, recording 6-DoF ground truth poses at 150 Hz. The marker-based cameras require at least four markers visible at all times. Thus, for three out of four glasses, five markers were used. For the only light HMD with less surface space to distribute markers on, four markers were attached.



(a) (b)

Figure 6.1: The hardware setup on the dashboard inside the car and the car itself used for the dataset recording. (a) There are three infrared cameras with flashers on top for the image recording (red). Two marker-based cameras acquire the 6-DoF ground truth poses (blue). (b) The dataset was recorded in a 7 series BMW, here on top with the utilized 360° camera (red) to acquire footage for simulation to be used during recording.

6.1.2 Calibration and Synchronization

A calibration to one common coordinate frame is required for an accurate and synchronous recording of the pose. We use the variables introduced in Chapter 3 for coherence. We obtained the extrinsics of each camera system T_1, T_2, and T_3 concerning the coordinate frame W through calibration. We receive the transformations TS_{W_T1}, TS_{W_T2}, and TS_{W_T3}. A static checkerboard inside the car is used for this purpose. The given points on the checkerboard and their relation to the common coordinate frame are known. The cameras are calibrated to the coordinate frame by localizing the same points in camera coordinates. The coordinate frame is set to the center of the car's frontal vehicle axle. The x-axis points towards the back of the car, y-axis to the right from the driver's perspective, and the z-axis points upwards. All cameras and glasses poses S are given with respect to this coordinate system. The glasses to car coordinate frame transformation is called TS_{W_S}. The various coordinate systems in the car are visualized in Figure 6.2.

The closest marker to the center of the respective glasses has been defined as the center of the local coordinate systems of the individual glasses model. In addition, a synchronized recording of the frames and poses is needed. The synchronization takes the current timestamp of a recorded image triple into consideration and computes the difference to the next pose timestamp. If the difference is zero, the pose obtained by the marker-based camera is accepted as the ground truth for the current frames. If there is no pose timestamp where the difference is zero, the next pose with a difference of up to 10ms difference is taken as the right pose. This

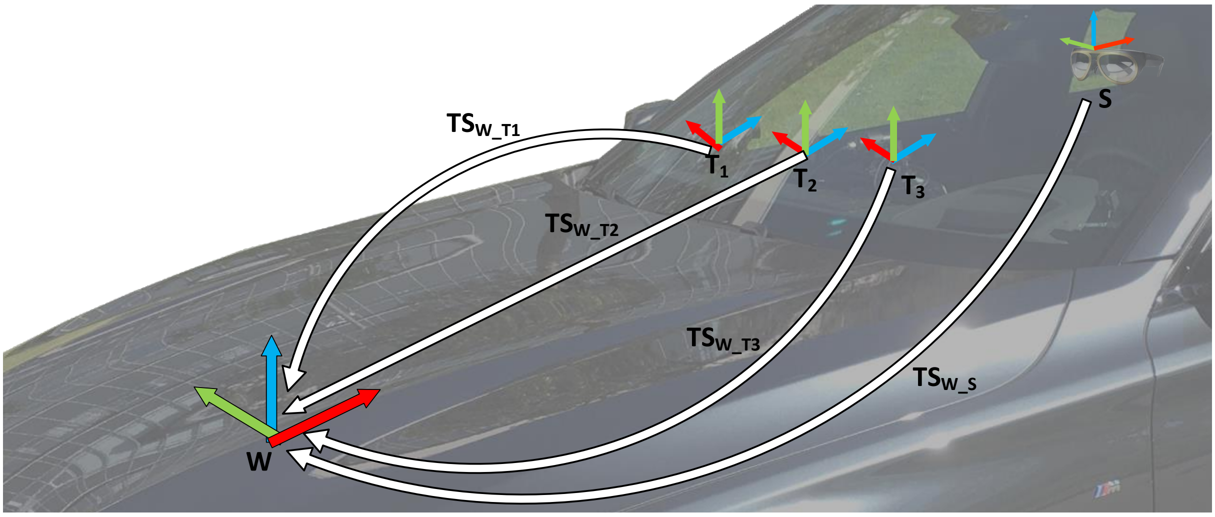


Figure 6.2: All coordinate systems and their transformations. The x,y, and z axes are colored red, green, and blue, respectively. The common coordinate frame W relies in the front of the car. The camera coordinates of three cameras T_1, T_2, and T_3 pointing towards the passenger and the glasses pose orientation S are highlighted.

is done one more time with up to a 20ms timestamp difference. If there is no pose to be found, the pose between the last and next pose is interpolated and adopted as the current pose. For the position, this is done through linear interpolation of the last and the next position. For the orientation, a spherical spline interpolation is performed.

6.1.3 Acquisition and Dataset Analysis

The 6-DoF pose of four different glasses in the reference coordinate frame and the three IR images are acquired synchronously (Figure 6.3).

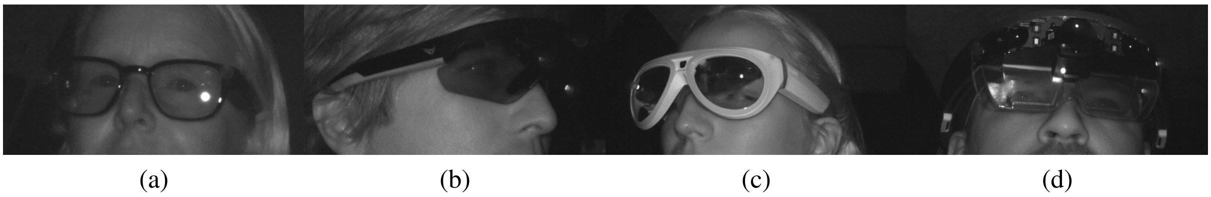


Figure 6.3: Example, cropped IR images of all four glasses in the recordings: (a) North Focals Generation 1, (b) Everysight Raptor, (c) Mini Augmented Vision (d) Microsoft Hololens 1.

The selected North Focals Gen 1 [101] acts as a more conventional-looking glasses. The representative for activity-related glasses is the Everysight Raptor glasses [102]. The Mini Augmented Vision glasses [103] represent all-day-use HMDs. The Microsoft Hololens 1 [104] was chosen as an industrial, larger, more powerful AR HMD. The recording has been conducted in a driver simulation. Our prototype car acted as a static simulation environment. We recorded a 1km long route with a 360° camera strapped on the given prototype. Based on this material, different angles are shown on three screens simultaneously, which were placed around the prototype car to trigger natural head movements (Figure 6.4).

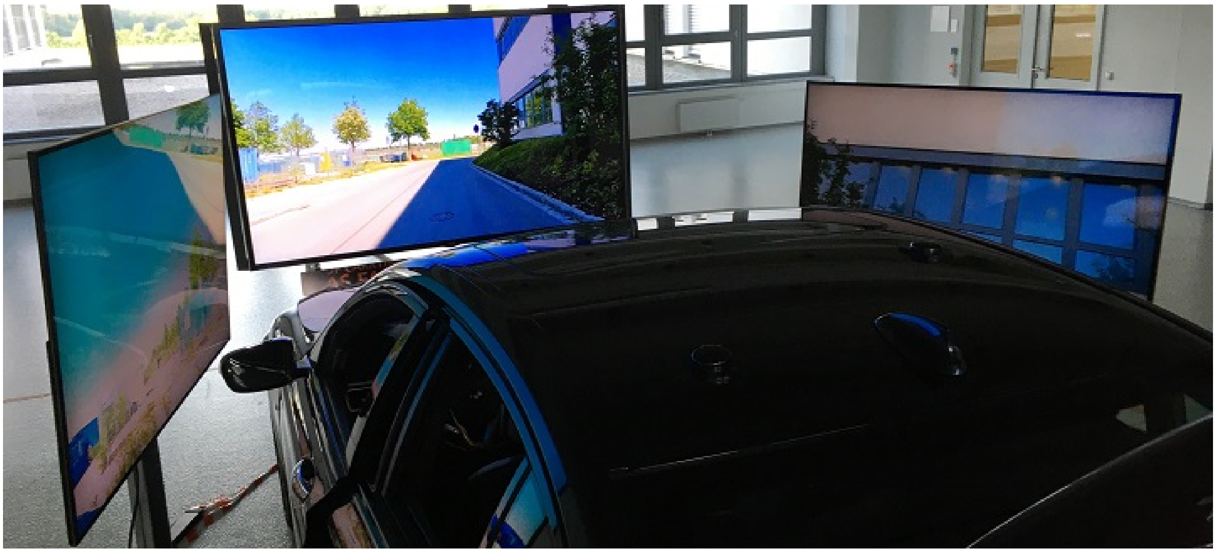


Figure 6.4: The simulation setup. A route is played which was prerecorded with 360° camera and edited accordingly for the three screens.

Four sequences for each HMD have been recorded with each participant, while the five-minute-long route was displayed on the screens. 14 subjects participated in the recording of the dataset. Multiple points of interest (POIs) were defined on this route, where the participants were instructed to look at (Table 6.1).

description	rotation	translation	other
driving	low	low	generic street observance
roundabout	med	med	observing roundabout, then looking for exit
intersection	high	med	looking left and right
POI	high	med	either left or right look
U-turn	high	med	left and right look

Table 6.1: The different situations on the route and occurring movements, ranked low, med or high. The last column provides additional information for each situation.

Between the instructed parts, the subjects had the freedom to either follow the road or look around. A certain bias in movement occurs with each ride through the defined route per subject.

To counterbalance the order for all HMDs, the balanced Latin Square method [105] has been used. The method generates a matrix with a balanced distribution of the position of an element in all rows and columns of a matrix. Four different matrices based on the four glasses have been generated. This way, the bias based on driving through the same route four times is minimized, as each participant wears the glasses in a different order.

The dataset has a 35.7% female and 64.3% male ratio. The mean age is 33.5 years, with a standard deviation of 10.73. Figure 6.5 displays the data distribution of the orientation on all axes. The orientation strongly suggests the frontal orientation during driving, visible on all axes.

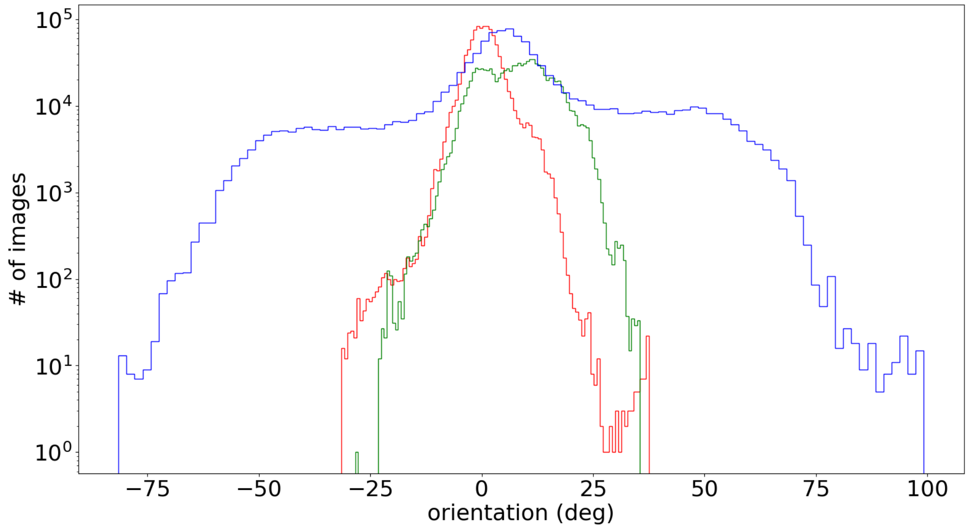
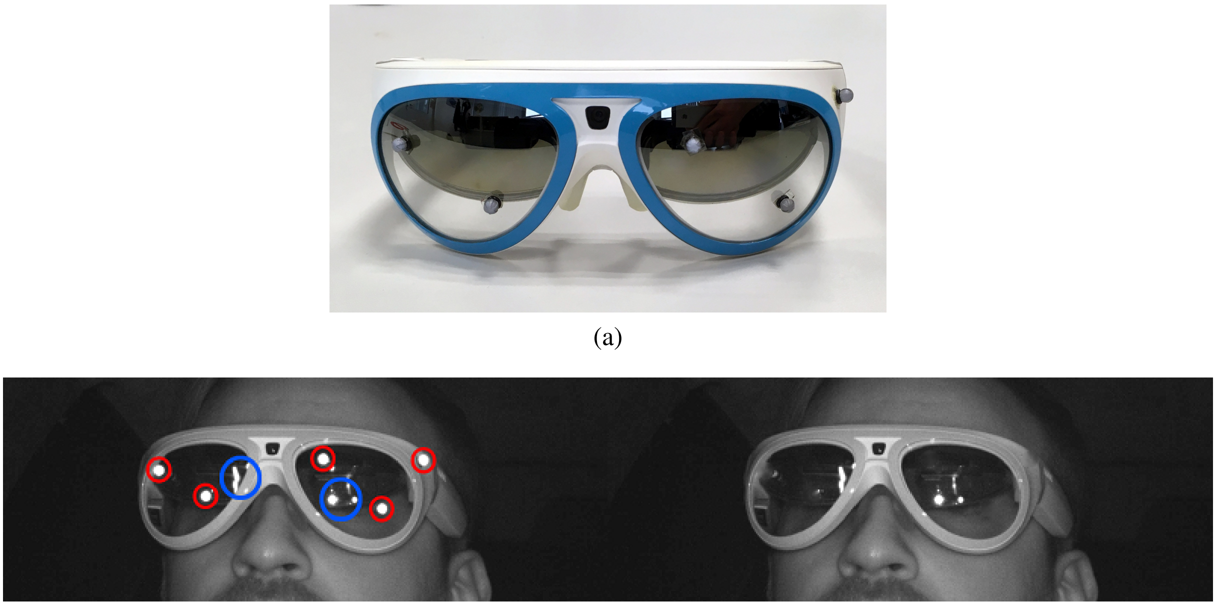


Figure 6.5: The glasses orientation distribution on all three individual angles in degrees. red: roll, green: pitch, blue: yaw.

6.1.4 Postprocessing

As the frames of the different glasses are recorded with attached markers to obtain an accurate ground truth pose, the markers on the glasses are visible on the recorded images. The markers are removed from the glasses in an automated process in postprocessing (Figure 6.6). This is done by utilizing the known marker positions, which the marker-based system can save. The algorithm combines them with the detection of bright circles. If bright circles can be detected on an image matching one of the given marker positions, the marker is removed. The OpenCV inpainting method based on the method of Telea [106] is used to hide the markers on the image.

The postprocessing concludes the HMDPose dataset acquisition. In summary, we captured infrared images from three different positions of 14 subjects, wearing four different AR glasses. The recording resulted in more than 3,000,000 infrared images enclosed with ground truth AR glasses pose.



(a)

(b) (c)

Figure 6.6: Example of the Mini Augmented Vision glasses before and after postprocessing. (a) Image of the glasses with the markers circled in red. (b) IR image before postprocessing, where markers are circled red and reflections are highlighted in blue. (b) After postprocessing without markers but visible reflections.

6.2 Head Pose Annotation Generation on the HMDPose Dataset

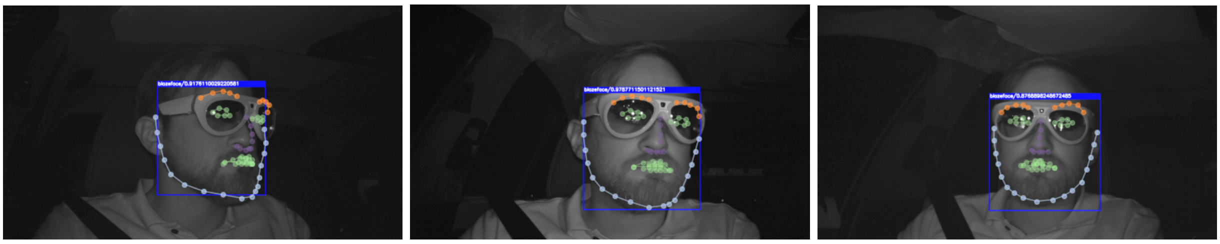
Another interesting analysis is applying pose algorithms on the head, even though the AR glasses are visible on the images. The aim of this analysis is the comparison with the AR glasses pose accuracy. Although the head pose always behaves differently from the AR glasses pose, if the head can be detected more accurately, it can be a decent supplementary approach to extend the AR glasses pose.

The only ground truth recorded for the dataset were AR glasses pose annotations. Thus, the generation of head pose labels on the same image data was required to enable a comparison. In this section, a pipeline is introduced for measuring head orientation and position based on the recorded triple images. The goal of the approach is to generate 6-DoF pose labels without using the ground truth annotations from the HMDPose dataset to exclude potential dependencies. For that, the triple IR images and the poses of the cameras in the reference coordinate system are used exclusively.

In the head pose annotation pipeline, facial landmarks are extracted first. They are used to compute their 3D positions in the reference frame. Subsequently, the head coordinate system is defined. Finally, the measured head poses are postprocessed.

6.2.1 Facial Landmarks Extraction & Landmark Reconstruction

In the first step, the head is detected, and its 2D bounding box is localized first in all cameras as an axis-aligned rectangle. The BlazeFace model [107] is then used to extract image features using the so-called Blazeblocks. Each block consists of depth-wise convolutional layers for speeding up the information flow. Afterwards, a regressor is used to obtain the final bounding box coordinates. After cropping the head using the predicted bounding box, a facial landmark extractor outputs a set of labeled facial keypoints in the form of prominent facial locations expressed in all pixel coordinate systems. Bulat et al. propose Face Alignment Network (FAN) [108] and show remarkable performance due to the beneficial use of the hierarchical parallel and multi-scale blocks [109]. FAN uses similar architecture as state-of-the-art models for human pose estimation. The network predicts 69 facial landmarks in total (Figure 6.7).



(a) Left View (b) Middle View (c) Right View

Figure 6.7: Visualization of the extracted facial landmarks. The blue box depicts the detected bounding box by the used head detector.

The extracted landmarks for all image triples of the dataset are then used for landmark reconstruction.

After extracting a set of facial landmarks per image in the image triples, the coordinates of 2D landmarks in the 3D space are determined. The reprojection estimation depicts a typical triangulation problem. The projection of a 3D point onto the pixel plane P_i of the camera i is non-linear. Therefore, the Bundle Adjustment (BA) technique is deployed to solve the triangulation problem. The estimated 3D points are reprojected onto the pixel coordinates of all three images of a triple image, and the reprojection error is minimized. The Levenberg–Marquardt non-linear optimizer [110] estimates the individual 3D point $\hat{\boldsymbol{P}}_i$ by minimizing a non-linear cost function:

$$\hat{\boldsymbol{P}}_i = \underset{\boldsymbol{P}_i}{\operatorname{argmin}} \sum_{k=1}^{3} \left\| \boldsymbol{p}_i^k - \boldsymbol{f}_k(\boldsymbol{P}_i) \right\|_2^2, \tag{6.1}$$

$\boldsymbol{p}_i^k$ are the 2D coordinates of a landmark i in the pixel frame k. $\boldsymbol{f}_k$ depicts the non-linear projection of 3D points onto the pixel frame k.

All N landmark points and all three images of a triple image are stacked into a single vector to perform a joint optimization, where the reprojection errors are minimized in all pixel frames.

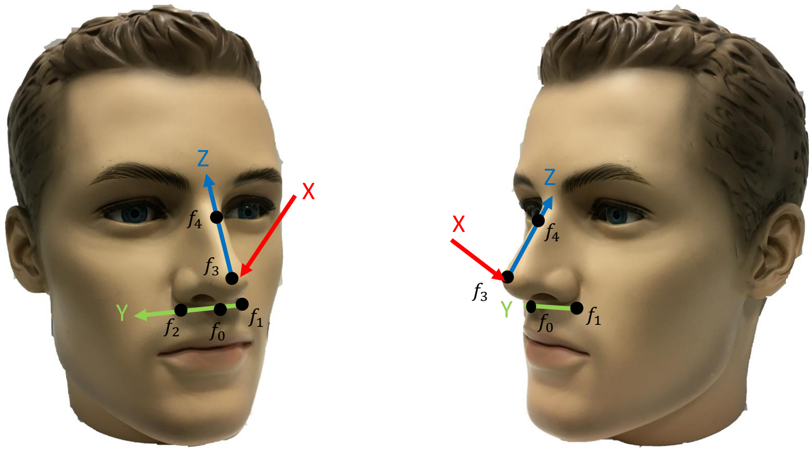
A non-linear least square estimation minimizes the norm of the bundled error vector:

$$(\hat{\boldsymbol{P}}_1, \ldots, \hat{\boldsymbol{P}}_N) = \underset{(\boldsymbol{P}_1,\ldots,\boldsymbol{P}_N)}{\operatorname{argmin}} \left\| \begin{matrix} \boldsymbol{p}_1^1 - f_1(\boldsymbol{P}_1) \\ \boldsymbol{p}_1^2 - f_2(\boldsymbol{P}_1) \\ \boldsymbol{p}_1^3 - f_3(\boldsymbol{P}_1) \\ \vdots \\ \boldsymbol{p}_N^1 - f_1(\boldsymbol{P}_N) \\ \boldsymbol{p}_N^2 - f_2(\boldsymbol{P}_N) \\ \boldsymbol{p}_N^3 - f_3(\boldsymbol{P}_N) \end{matrix} \right\|_2^2 . \tag{6.2}$$

The iterative solver of this optimization follows a particular update policy. The 3D positions of the landmarks are initialized using the results of a linear triangulation for two randomly selected cameras. The optimal 3D facial landmarks in the reference frame are derived after a number of update iterations.

6.2.2 Head Pose Computation and Postprocessing

The definition of the head pose is related to the definition of the local head coordinate system. For every single frame of the dataset, the center of the head coordinate system and its orthonormal basis is defined by means of the computed 3D facial landmark positions as presented in Figure 6.8.



(a) Side View (b) Front View

Figure 6.8: Definition of the head coordinate system shown from two different views.

Points of a rigid region of the head are chosen for this purpose. Inspired by Breitenstein et al. [111], the nose is chosen as our region of interest. The subnasale key-point f_0 is defined as the center of the head, as it is always visible in at least two cameras. The Y-axis is defined as the normalized vector pointing from the left alare f_1 to the right alare f_2 feature. The Z-axis is defined by the normalized vector linking the nose tip f_3 and the mid-endocanthion point f_4 of the face. The X-axis is given by the cross-product of the axes Y and Z.

To eliminate outliers in the generated labels, the annotations of the dataset are postprocessed. After computing the head pose for every frame independently, the poses are considered as time-series data. MAD-based outlier detector [112] eliminates the outliers. The MAD scores for every single pose in all dimensions are computed as follows [113]:

$$\boldsymbol{M}_i = \frac{c \cdot (x_i - \tilde{x})}{median\{|x_i - \tilde{x}|\}} . \tag{6.3}$$

$\tilde{x}$ is the sample median, and the denominator represents the MAD. The constant c is set to 0.6745 as defined by Garcia [113]. If the score $\boldsymbol{M}_i$ of a sample is larger than a threshold D, it is considered an anomaly. The anomalies are replaced by averaging the pose of the previous and next frames. A low pass filter is used to smooth the poses. Finally, a first-order Butterworth filter is used to smooth the 6-DoF poses.

6.2.3 Comparison of Head Pose and AR Glasses Pose Labels

To validate the quality of the generated head pose labels, a comparison with the AR glasses ground truth is needed.

Figure 6.9 depicts the data distribution on the same sequence of the head pose and the HMD pose labels, which confirm the previous observations. The distance between the centers of the corresponding coordinate frames and their initial orientation is observable.

For a more detailed comparison, the differences of the label sets are quantified. The pose of the HMD is defined as $\boldsymbol{p}_t^{S,HMD}$, and the pose of the head as $\boldsymbol{p}_t^{S,Head}$ for frame t from the sequence S. For comparison, three types of deviations for every single sequence S:

- Inter-Objects deviation computes the spatial difference between the head and glasses pose of a single frame t. It provides insight about the relationship between the head and the glasses. This deviation measure is biased with lower variance. Hence, the overall translational and rotational shifts between head and glasses are given by the bias, while the amount of variation over time is determined by the variance. The Inter-Objects deviation is mathematically defined as follows:

$$InterDev_{Head}^{HMD}(t) = dev(\boldsymbol{p}_t^{S,Head}, \boldsymbol{p}_t^{S,HMD}) . \tag{6.4}$$

- Intra-Object deviation computes the temporal evolution of the pose in a sequence S from either the head or HMD labels. The Intra-Object deviation is defined as follows:

$$IntraDev_{Obj}(t) = dev(\boldsymbol{p}_t^{S,Obj}, \boldsymbol{p}_{t-1}^{S,Obj}), \quad Obj \in \{Head, HMD\} . \tag{6.5}$$

- Inter/Intra-Object deviation computes the Inter-Object of the Intra-Object deviation. It describes the Spatio-temporal evolution of the pose in a sequence S relative to a pose

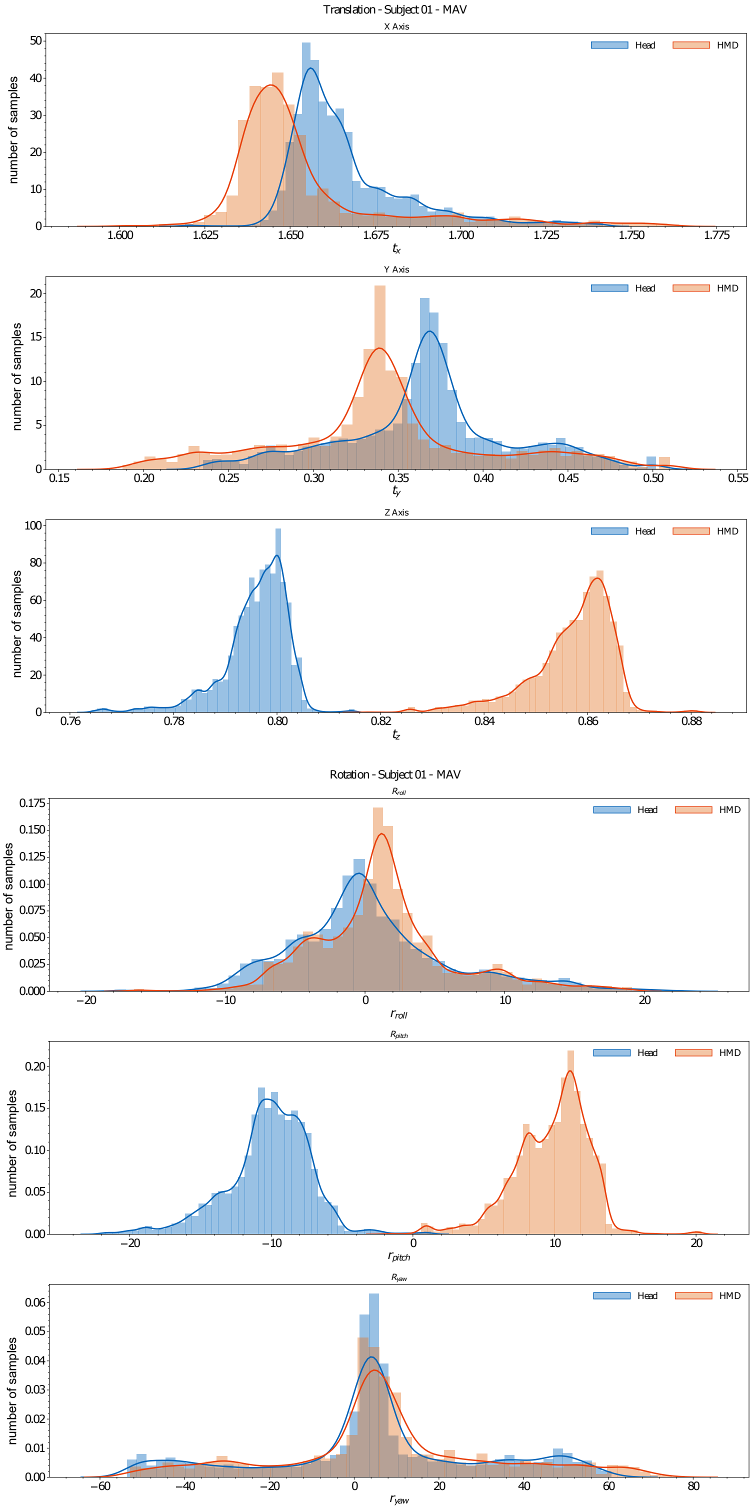


Figure 6.9: Distributions of the translation and rotation of a sequence in Euclidean and Euler spaces, respectively.

of another object, thus measures the distance between the dynamics of the head and AR glasses labels. The Inter/Intra-Object deviation is defined as:

$$InterIntraDev_{Head}^{HMD}(t) = dev(IntraDev_{Head}(t), IntraDev_{HMD}(t)) \,. \tag{6.6}$$

The variable dev can be any deviation metric. In the following, the mean absolute error (MAE), the root mean squared error (RMSE), and the standard deviation (STD) are used. Table 6.2 shows the Inter-Object deviation between the head and the glasses.

	Translation				Orientation			
Metric	X	Y	Z	Avg	Roll	Pitch	Yaw	Avg
MAE	18.20	25.61	54.20	32.67	1.75	10.28	4.48	5.51
RMSE	22.61	29.81	57.72	36.71	2.27	12.85	6.27	7.13
STD	13.41	15.25	19.85	16.17	1.44	7.71	4.39	4.51

Table 6.2: Inter-Objects translation and rotation deviations between HMD and head labels.

A deviation of almost 1.8cm, 2.5cm, and 5.5cm repetitively along the X, Y, and Z-axes are noticeable due to the shifted glasses and the head centers. The standard deviation shows that the deviation for the translation and the rotation is not time-invariant. This observation confirms that the head pose differs from the HMD pose, while the difference is not constant.

Tables 6.3 and 6.4 display the dynamics in the evolution of the poses.

	Translation				Orientation			
Metric	X	Y	Z	Avg	Roll	Pitch	Yaw	Avg
MAE	0.31	0.65	0.10	0.36	0.04	0.07	0.28	0.13
RMSE	0.73	1.41	0.22	0.79	0.09	0.26	0.62	0.32
STD	0.66	1.25	0.19	0.70	0.07	0.24	0.55	0.29

Table 6.3: Intra-Object translation and rotation deviations for the HMD labels.

	Translation				Orientation			
Metric	X	Y	Z	Avg	Roll	Pitch	Yaw	Avg
MAE	0.19	0.53	0.06	0.26	0.05	0.05	0.25	0.11
RMSE	0.36	0.94	0.12	0.47	0.07	0.08	0.42	0.19
STD	0.30	0.77	0.10	0.39	0.06	0.06	0.34	0.15

Table 6.4: Intra-Object translation and rotation deviations for the head labels.

The tables show strong similarities between the metric values, as the HMD and head movements are synchronized. The minor difference stems from two different effects. First, the inconstant positions of the HMD on the head affect the pose, resulting in varying poses. Second, the definition of the head coordinate system based on facial landmarks is affected by the mimic of the subject. Even though relatively static facial landmarks are selected for later tracking, the mimic of the subject can still affect them. Hence, they are not as constant as markers on AR glasses.

Table 6.5 presents the Inter/Intra-Object deviation. It reveals the similarity between the dynamics of the head and AR glasses labels.

	Translation				Orientation			
Metric	X	Y	Z	Avg	Roll	Pitch	Yaw	Avg
MAE	0.21	0.38	0.06	0.20	0.05	0.08	0.19	0.10
RMSE	0.51	0.75	0.14	0.47	0.10	0.25	0.34	0.23
STD	0.47	0.67	0.12	0.42	0.08	0.23	0.28	0.20

Table 6.5: Inter/Intra-Object translation and rotation deviations between HMD and head labels.

A strong correlation between the evolution of the HMD pose and the head pose can be observed. Nonetheless, the poses of both objects differ.
In the following subsection, the complete head pose label set regarding its statistical distribution of the position and orientation is compared to the initial HMDPose labels.

6.2.4 Data Analysis

The structure of both annotation sets is similar. This can also be observed in their data distributions. The distribution for the HMD annotations can be seen in Figure 6.10, whereas the 6-DoF head pose distribution is shown in Figure 6.11.

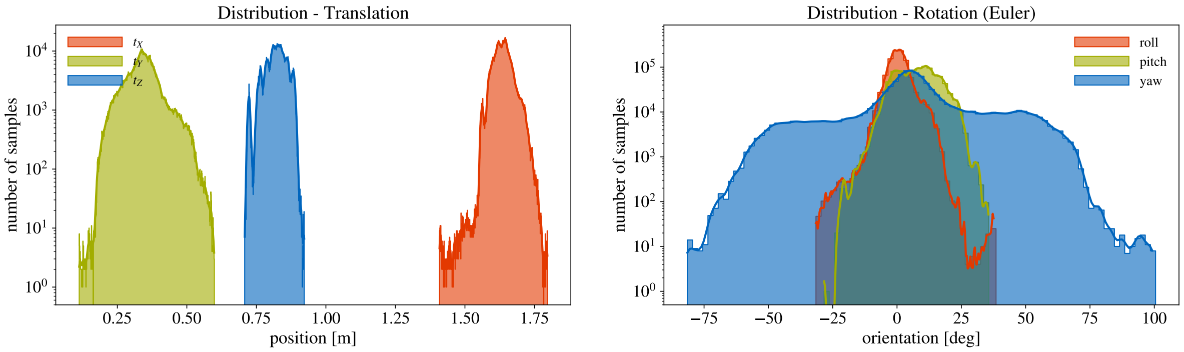


Figure 6.10: Position and orientation distribution per axis of the initial HMDPose labels.

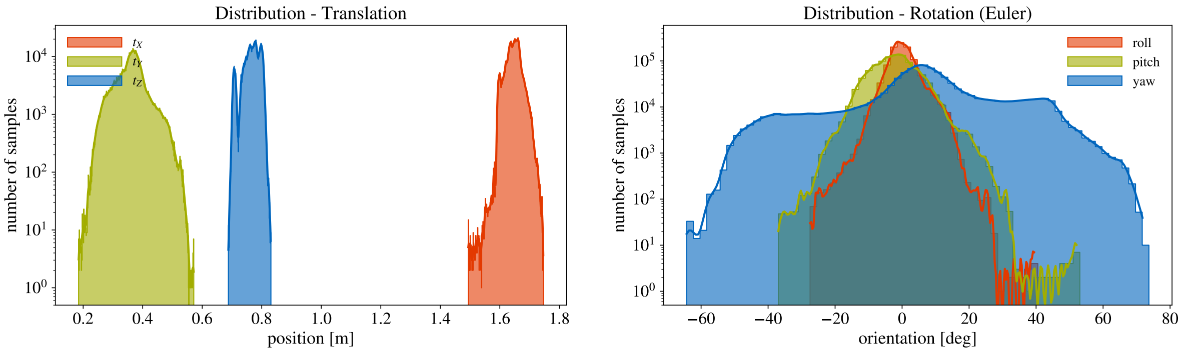


Figure 6.11: Position and orientation distribution per axis of the generated head pose labels.

The shape of the distributions is similar and centered to zero. The head pose annotations have a smaller range on the yaw but a wider pitch distribution compared to the HMD labels, as it can be seen in Table 6.6 and Table 6.7.

	Translation			Orientation		
Metric	X	Y	Z	Roll	Pitch	Yaw
Mean	1.640	0.342	0.822	0.735	7.870	7.217
Std	0.030	0.059	0.035	4.170	7.593	22.915

Table 6.6: Translation and rotation statistics of the HMDPose dataset.

	Translation			Orientation		
Metric	X	Y	Z	Roll	Pitch	Yaw
Mean	1.640	0.342	0.822	0.164	-2.118	7.345
Std	0.030	0.059	0.035	4.128	6.666	19.997

Table 6.7: Translation and rotation statistics of the generated head pose labels.

The position distributions are identical. The movement area in the car is restricted during driving, which leads to the same distribution, clearly visible in the tables.

In conclusion, we observe strong similarities between the two label sets. This points towards a reliable result of the head pose label generation pipeline, despite the partial occlusion of the face by the AR glasses. The next chapter evaluates several HMD and head pose estimators trained on both label sets.

6.3 Summary

This chapter introduced and described our HMDPose dataset. Firstly, the hardware setup and required calibration and synchronization steps are presented. The setup included a real car with cameras on the dashboard, capturing infrared images and tracking markers attached to the AR glasses. The camera and tracking hardware inside the car require synchronization among them and must be calibrated to the car coordinate frame. Then we describe the acquisition, which included four different AR glasses and 14 different subjects. As the ground truth labels were collected through a marker-based solution, we later removed the markers from the images.

Based on this data collection process, we generated head pose labels. For this, a pipeline extracting facial landmarks, reconstructing their 3D position, and a later head pose computation was necessary. In a consecutive postprocessing step, the labels were cleaned and filtered. The chapter ends with an evaluation of the head pose labels and a comparison with the AR glasses labels.

The dataset, including the HMD and head labels, was used to train a variety of networks. The following chapter will evaluate all trained networks introduced in Chapters 4 and 5 regarding HMD and head pose estimation performance. We will also compare the performance of HMD and head pose estimation.

7 Evaluation of Head and AR Glasses Pose Estimation

In previous Chapters 4 and 5, we presented various AR glasses and head pose estimation techniques and fusion methods for potential combinations of the approaches. Subsequently, we presented our datasets in Chapter 6, which includes triple infrared images of several persons, appended with HMD and head pose annotations.

Finally, in this chapter, the introduced methods will be evaluated. First, we present the evaluation metrics utilized for benchmarking. Then, in Section 7.2, we evaluate head orientation estimation methods, which we initially developed for the AutoPOSE dataset [95]. The insights of this evaluation about infrared-based pose estimation were used to build the AR glasses pose estimators.

In Section 7.3, we benchmark our image-based feed-forward pose estimation methods, image-based recurrent pose trackers, and point cloud-based pose estimation methods.

We then assess a selection of them on head pose labels in Section 7.4. We compare them to their AR glasses label trained variants. The comparison aims for their general estimation performance and generalization ability.

The following Section 7.5 analyzes the fusion methods of several HMD pose estimators as well as fusing head and HMD pose estimation. We conclude the chapter with Section 7.6, which conducts a real-world evaluation of the methods.

7.1 Evaluation Metrics

For an insightful comparison of the methods, we define several evaluation metrics. We consider three different metrics.

At first, we define the Mean Absolute Error (MAE), also known as the angle estimation error.

$$MAE := \frac{1}{n}\sum_{i=1}^{n} |y - \widetilde{y}| \tag{7.1}$$

We calculate it on each individual axis and on all axes combined to obtain the total estimation error.

Secondly, we use the Root Mean Squared Error (RMSE) to penalize significant errors in the estimation larger. Thus high estimation variations lead to higher errors.

$$RMSE := \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-\widetilde{y})^2} \tag{7.2}$$

Lastly, we utilize the Balanced Mean Angular Error (BMAE). It takes the unbalanced amount of different head orientations due to driving and their bias towards frontal orientation into account by defining different ranges:

$$BMAE := \frac{d}{k}\sum_{i=1}\phi_{i,i+d}, i \in d\mathbb{N} \cap [0,k], \tag{7.3}$$

$\phi_{i,i+d}$ is defined the average angular error. For our evaluation, we set the section size d to 5 degrees and the range size k to 180 degrees.

For the position error, we utilize the L_2 loss. We calculate each metric on each individual axis and on all axes combined.

7.2 Head Orientation Estimation Evaluation on the AutoPOSE Dataset

Section 5.1 presented multiple head orientation regression techniques, including the POSEidon model [71, 96], the HPN model [70], and our HON and ResNetHG networks. We train all mentioned Deep Neural Networks on the 64×64 and 128×128 pixel cropped images of the dataset. An exception is HON, which was specifically designed for the 128×128 pixel images. Figure 7.1 gives an overview of the evaluation pipeline. The models' training gave us insight into what type of architectures are more suitable for IR images on different resolution levels as we trained on 64×64 and 128×128 pixel images.

Network Size Evaluation

We also evaluate different network depths on ResNetHG and the HON model to deduce what amount of layers for IR image-based head pose regression is more fitting. After comparing ResNet-50, ResNet-34, and ResNet-18 as a basis, including the added skip connections, ResNet-18 showed the most promising results for the task and data at hand. In general, we found out that the more layer we have in the ResNetHG and HON, the worse the estimation becomes. This is illustrated in Figure 7.2, where we plot the error in degree on three metrics of the variants mentioned earlier of the HON and the ResNetHG architecture. Both results may be caused due to the IR images being grayscale and having one channel. Compared to RGB images that usually have 24-bit information, grayscale images only have 8-bit information, leading to overparametrization in deeper networks. Therefore, smaller networks may be sufficient for IR images. Thus, we settled on our models with fewer layers for ResNetHG and HON.

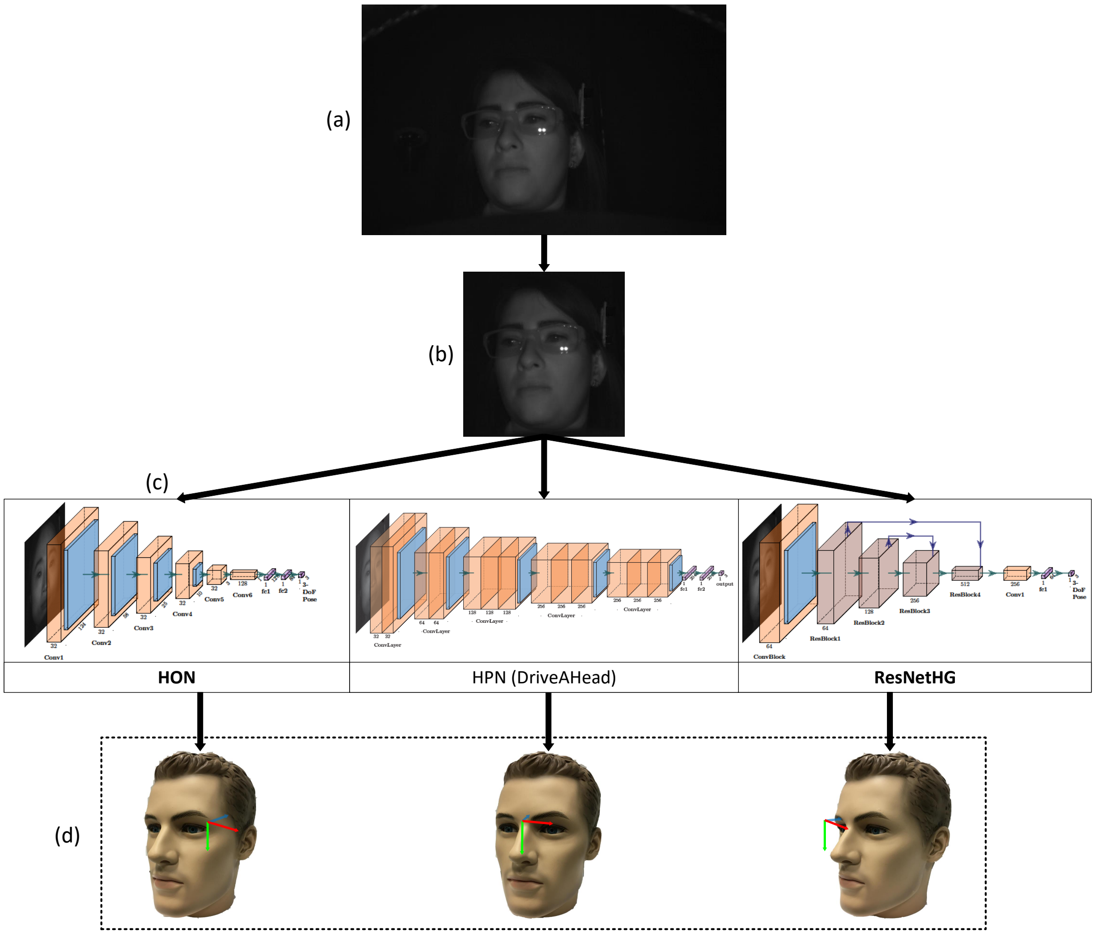
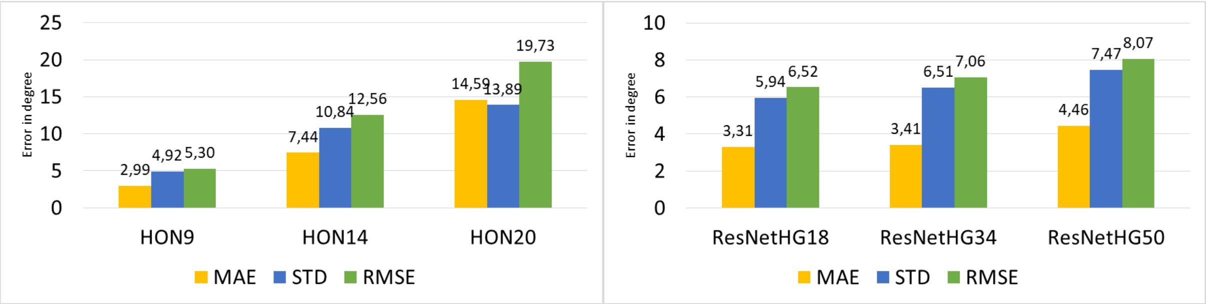


Figure 7.1: Evaluation on the AutoPOSE dataset. (a) On top is an example frame from the AutoPOSE dataset. (b) Then, we crop the images to head size and a resolution of 128 × 128 pixel images. Additionally, we generate 64 × 64 pixel images. (c) In a next step, the networks regress the head orientation. The architectures used on the 128 × 128 pixel images are shown. (d) The outcome is further evaluated on different metrics.

Orientation Estimation Results

At first, we compare the results of our trained networks on 64 × 64 cropped images (Table 7.1). We additionally report the Standard Deviation (STD). The results show that the training of the POSEidon network on the AutoPOSE dataset performs best on the 64-pixel images, being the smallest network. ResNetHG18 achieves comparable results on the MAE, STD, and RMSE, especially regarding Pitch and Roll. The BMAE shows worse results, suggesting a difference in performance on different orientation ranges of ResNetHG18.

With our 128 × 128 pixel-cropped images, we achieve lower error on every metric (Table 7.2). The figure shows that our HON achieves the best results on all metrics and axis, excluding the roll on MAE, where ResNetHG18 produces less error. ResNetHG18 performs comparable to



(a) (b)

Figure 7.2: Comparison by layer amount for the ResNetHG and HON on average error on three metrics. (a) compares HON with 9, 14 and 20 layers, (b) shows ResNetHG with ResNet-18, ResNet-34 and ResNet-50 as a basis.

Metric	Model	Pitch	Roll	Yaw	Avg
MAE	POSEidon [96]	**2.96**	**3.16**	**3.99**	**3.37**
	ResNetHG18	4.02	3.32	5.20	4.18
	HPN (DriveAHead) [91]	8.18	6.68	13.31	9.39
STD	POSEidon [96]	**4.63**	**3.93**	**7.82**	**5.46**
	ResNetHG18	6.25	4.98	11.57	7.60
	HPN (DriveAHead) [91]	10.36	9.62	21.68	13.89
RMSE	POSEidon [96]	**4.73**	**4.55**	**7.98**	**5.97**
	ResNetHG18	6.37	5.20	11.58	8.20
	HPN (DriveAHead) [91]	11.25	9.70	21.69	15.18
BMAE	POSEidon [96]	**7.10**	**9.42**	**19.05**	**11.86**
	ResNetHG18	12.18	13.58	35.41	20.39
	HPN (DriveAHead) [91]	20.96	23.66	59.69	34.77

Table 7.1: Results on the 64×64 pixel cropped images of the AutoPOSE dataset of the POSEidon model [96], our ResNetHG18 network, and the HPN model [91]. The lowest values per axis are highlighted.

HON on many metrics and axis. We observe a performance loss on the yaw on STD, RMSE, and BMAE. The HPN model leads to considerably higher error rates than other methods amongst all axes, metrics, and resolutions.

We finally compare the resolution levels and the model performances. Since the POSEidon model was trained only on 64×64 pixel images and HON only on 128×128 pixel images, we directly compare them as they are both less deep networks for each resolution level (Table 7.3). We can observe that HPN performs similarly on both resolution levels, having the highest error compared to the other networks. For our ResNetHG18 architecture, we see a performance boost

Metric	Model	Pitch	Roll	Yaw	Avg
MAE	HON	**2.68**	2.73	**3.56**	**2.99**
	ResNetHG18	3.29	**2.48**	4.15	3.30
	HPN (DriveAHead) [91]	8.32	6.87	13.81	9.67
STD	HON	**4.21**	**3.55**	**6.99**	**4.92**
	ResNetHG18	4.86	3.74	9.23	5.94
	HPN (DriveAHead) [91]	10.36	9.62	21.68	13.89
RMSE	HON	**4.25**	**3.87**	**7.15**	**5.30**
	ResNetHG18	4.98	3.98	9.33	6.52
	HPN (DriveAHead) [91]	11.38	9.81	21.76	15.27
BMAE	HON	**4.97**	**7.26**	**15.10**	**9.11**
	ResNetHG18	8.00	8.18	27.62	14.60
	HPN (DriveAHead) [91]	20.93	23.96	59.4	34.81

Table 7.2: Results on the 128×128 pixel cropped images of the AutoPOSE dataset of our HON model, our ResNetHG18 model, and the HPN model [91]. The lowest values per metric and axis are highlighted.

Resolution	Model	MAE	STD	RMSE	BMAE
64×64	POSEidon [96]	3.37	5.46	5.97	11.86
128×128	HON	**2.99**	**4.92**	**5.30**	**9.11**
64×64	ResNetHG18	4,18	7.60	8.20	20.39
128×128	ResNetHG18	3.30	5.94	6.52	14,60
64×64	HPN (DriveAHead) [91]	9.39	13.89	15.18	34.77
128×128	HPN (DriveAHead) [91]	9.67	13.89	15.27	34.81

Table 7.3: Direct comparison of all methods on 64×64 and 128×128 resolutions. The lowest values per metric are highlighted.

on the higher resolution level as the error declines on all metrics. Especially the error on various degree levels as measured by BMAE and outliers as measured in RMSE declined significantly. Furthermore, compared to the POSEidon result provided in the dataset paper, our HON model performs better on all metrics.

On a direct comparison of the models on different resolutions, we demonstrate that HON outperforms all other methods, achieving less than 3 degree error on average on the MAE. Our ResNetHG18 model achieves comparable but less accurate results than its VGG-based CNN counterparts like POSEidon and HON with fewer layers. As we concluded from our ablation study of different depths for our HON and ResNetHG architectures, less layers result in better estimation performance. Thus, the better performance of the CNNs may be caused due to the ResNet-18 foundation of ResNetHG18, containing considerably more layers and parameters.

Discussion

Our evaluation on the AutoPOSE dataset proved that deep neural networks with fewer layers perform better on head orientation regression based on IR images. In addition, we have shown in our experiments on the effects of resolution that higher resolution images lead to lower estimation errors. Finally, our HON and ResNetHG18 architectures outperform all other methods.

Based on the results, we deduced some aspects, which we further used in all other networks. Our AR glasses and head pose estimation networks on the HMDPose dataset are CNNs with a comparably small amount of layers. In case we deploy residual neural networks as a backbone like ResNet, we deploy ResNet-18. We also aim for higher input resolutions.

7.3 AR Glasses Pose Estimation Evaluation

Building upon the head orientation analysis results on the AutoPOSE dataset, we developed a variety of CNNs for 6-DoF pose estimation, as presented in Section 4. At first, we evaluate the image-based feed-forward pose estimation algorithms, trained on the images of the HMDPose dataset with AR glasses labels.

7.3.1 Feed-Forward Pose Estimation Results

For our evaluation, we use the full images of the HMDPose dataset, which contains 1 million infrared image triples, with resolutions of 1280×752 pixel images. We downscale them to 320×188 pixel as input. We aim to have a high resolution, improving the estimation accuracy as shown in the previous section. At the same time, too large images require very deep networks, which require longer trainings and do not show improvements for the task, as also shown in the head orientation regression.

Our training, validation, and test split is 94/3/3. The split is based on the widely used 98/1/1 split for Big Data. We slightly adjust the split given the high framerate of 60FPS in the HMDPose dataset. We shuffle the data before splitting.

Single vs. Multi-View Comparison

In a first analysis, we present a study on single and multi-view image-based AR glasses pose estimation with our GlassPose and GlassPoseRN methods (Figure 7.3). We train and evaluate the two custom-developed glasses pose estimation networks with one, two, and three input images on the HMDPose dataset. The comparison aims to identify the influence of the number of input images between single-view and multi-view Deep Learning-based object pose estimation approaches. This information is crucial for real-world deployment, as car vendors aim to keep

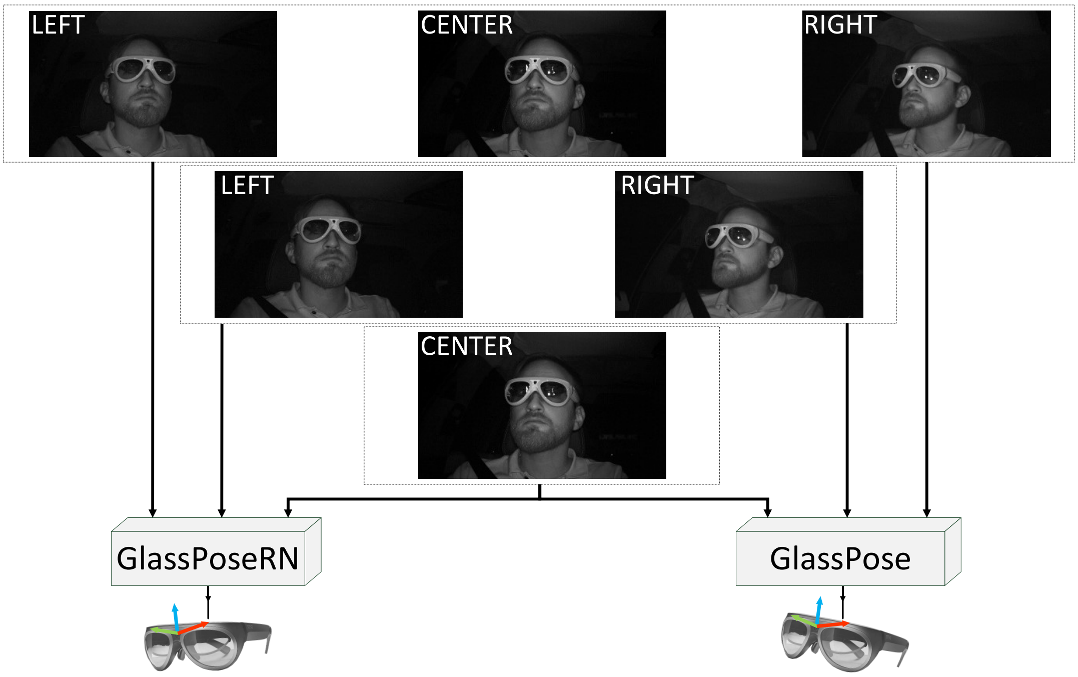


Figure 7.3: Overview of the three different view cases compared on GlassPose and GlassPoseRN for AR glasses pose estimation.

the number of additional hardware built in a car as low as possible. Hence, the affect of the number of views on the pose estimation quality is critical.

Table 7.4 shows the orientation error of our GlassPose and GlassPoseRN methods on the defined metrics on all axis individually and on average for all three view combinations. GlassPose shows results between 0.60° and 0.69° for the MAE with a minor change between the number of views. The RMSE is higher for stereo, resulting in 1.13° on average compared to 1.02° for the other view variants. Similarly, the BMAE for the stereo model is 4.29°, while being 4.03° and 3.89° for the one and three-view-model, respectively. GlassPoseRN performs significantly better with even fewer differences between the number of views. The MAE and the RMSE are 0.10° and 0.19° for all different view combinations on average, respectively. The BMAE is minimally higher on the three view variant with 0.24° against 0.18° for the other view combinations on all axes combined. Table 7.5 shows the positional L_2 error of the GlassPose and GlassPoseRN methods on all individual axes and in total for all three view combinations. The position estimation minimally improves by considering more than one view. For GlassPose, the overall L_2 error is 5.65mm for one view compared to 4.89mm and 5.43mm for two and three views, respectively. GlassPoseRN results in a generally lower error, where the L_2 error drops from 1.09mm for one view to 0.94mm and 0.90mm for two and three views.

Our experiments reveal little difference for orientation estimation with an increasing number of views. For position, we observe some decrease in error with more views. In general, for a more cost-efficient setup, one image can be used as it results in a comparably similar error. If the

		GlassPose				GlassPoseRN			
# Views	Metric	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg
1	MAE	**0.65**	0.60	**0.60**	**0.62**	0.07	**0.10**	0.14	0.10
	RMSE	**1.02**	**0.97**	1.08	**1.02**	0.10	**0.24**	0.24	0.19
	BMAE	4.64	1.76	5.71	4.03	0.20	**0.11**	**0.24**	**0.18**
2	MAE	0.69	0.60	0.61	0.63	0.07	**0.10**	**0.13**	0.10
	RMSE	1.23	0.99	1.17	1.13	0.10	**0.24**	**0.22**	0.19
	BMAE	5.40	1.92	5.55	4.29	**0.18**	0.12	**0.24**	**0.18**
3	MAE	0.67	0.60	0.61	0.63	**0.06**	0.11	0.14	0.10
	RMSE	1.06	0.98	**1.03**	**1.02**	**0.09**	0.25	**0.22**	0.19
	BMAE	**4.44**	**1.69**	**5.54**	**3.89**	0.29	0.12	0.32	0.24

Table 7.4: Orientation results of the GlassPose and GlassPoseRN approaches by view combinations on the given error metrics for the roll, pitch, yaw and the average in degrees. The lowest values per view for each approach are highlighted.

	GlassPose				GlassPoseRN			
# Views	x	y	z	L_2	x	y	z	L_2
1	2.67	3.22	2.56	5.65	0.66	0.54	0.42	1.09
2	**2.63**	**2.37**	**2.28**	**4.89**	0.59	**0.46**	**0.35**	0.94
3	2.85	2.73	2.54	5.43	**0.49**	0.50	**0.35**	**0.90**

Table 7.5: Results for the positional, Euclidean error of the GlassPose and GlassPoseRN approaches in millimeters. The lowest values per view for each approach are highlighted.

main goal is a minimal pose estimation error where the setup cost is not the focus, more images can bring some improvements. We proceed with evaluations on triple images.

Triple Image-based Feed-Forward Approach Comparison

Regarding image-based feed-forward pose estimation, we developed three different methods: GlassPose, GlassPoseRN, and CapsPose. In addition, we benchmarked a state-of-the-art, Regression-via-Classification approach named RandomizedBins, which we extended for position estimation, as described in Subsection 4.1.4. We compare the four approaches regarding orientation and position estimation in the following.

Orientation Results On Objects Per Method Table 7.6 shows an overview of the four approaches regarding their orientation estimation performance, trained and tested on the 94/3/3 data split for all individual glass models and combined.

Glasses-		RandomizedBins[81]				GlassPose				GlassPoseRN				CapsPose			
Type	Metric	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg
EVS	MAE	4.34	3.96	21.40	9.90	0.55	0.50	0.51	0.52	0.07	0.08	0.15	0.10	**0.05**	**0.06**	**0.12**	**0.08**
	RMSE	5.28	4.95	26.42	12.22	0.86	0.80	0.84	0.83	0.10	0.11	0.24	0.15	**0.07**	**0.08**	**0.20**	**0.12**
	BMAE	16.79	10.19	35.78	20.92	4.21	1.72	1.45	2.46	0.29	0.13	0.26	0.22	**0.07**	**0.08**	**0.20**	**0.12**
MAV	MAE	5.30	8.60	20.24	11.38	0.51	0.55	0.44	0.50	0.07	**0.08**	0.15	0.10	**0.06**	**0.08**	**0.12**	**0.09**
	RMSE	6.07	9.65	25.88	13.87	0.84	0.82	0.70	0.79	**0.09**	**0.10**	0.23	0.14	**0.09**	0.11	**0.20**	**0.13**
	BMAE	14.09	12.11	33.56	19.92	2.29	1.43	1.37	1.70	0.25	**0.11**	**0.20**	0.19	**0.09**	**0.11**	**0.20**	**0.13**
HOLO	MAE	10.41	5.52	15.46	10.46	0.63	0.53	0.52	0.56	**0.06**	0.08	0.14	0.09	**0.06**	**0.06**	**0.12**	**0.08**
	RMSE	11.02	7.22	22.53	13.59	0.92	0.83	0.82	0.86	0.09	0.11	0.22	0.14	**0.08**	**0.08**	**0.20**	**0.12**
	BMAE	14.81	13.51	35.65	21.32	2.51	2.27	1.51	2.10	0.16	0.09	0.18	0.15	**0.08**	**0.08**	**0.17**	**0.11**
NORTH	MAE	4.40	7.93	15.97	9.44	0.56	0.56	0.52	0.55	0.08	**0.18**	0.17	0.15	**0.06**	**0.18**	**0.14**	**0.13**
	RMSE	5.30	9.89	23.17	12.79	0.90	1.01	0.84	0.92	0.11	**0.48**	0.27	0.29	**0.09**	0.50	**0.23**	**0.27**
	BMAE	11.87	20.19	39.80	23.95	2.66	3.35	2.51	2.84	0.22	0.27	0.30	0.26	**0.12**	**0.20**	**0.21**	**0.18**
ALL	MAE	3.43	10.36	17.87	10.55	0.67	0.60	0.61	0.63	0.06	0.10	0.14	0.10	**0.05**	**0.09**	**0.12**	**0.09**
	RMSE	4.46	12.41	24.55	13.81	1.06	0.98	1.03	1.02	0.09	0.25	0.22	0.19	**0.08**	**0.24**	**0.21**	**0.18**
	BMAE	17.17	18.81	48.49	28.15	4.44	1.69	5.54	3.89	0.29	0.12	0.31	0.24	**0.09**	**0.09**	**0.21**	**0.13**

Table 7.6: Orientation results of the GlassPose, GlassPoseRN, and CapsPose approaches compared to RandomizedBins on the given error metrics in degrees. The Everysight Raptor is referenced as EVS, Hololens 1 as HOLO, North Focal Generation 1 as NORTH and the Mini Augmented Vision glasses as MAV. ALL stands for all glasses combined. The roll, pitch, yaw and the average of all three axes are given on the defined metrics. The lowest values per glasses model are highlighted.

Randomized Bins shows similar errors on the individual glasses, while ALL shows errors mainly on the higher end. RandomizedBins performs best on MAE for EVS and NORTH, with an average MAE of 9.90° and 9.41°, both being the smaller glasses types in the dataset. All other objects individually and combined result in similar MAE averages. On the yaw, a considerably high error among all metrics and all objects is observable, which is the heading angle of the glasses. The BMAE is significantly higher for ALL than individual glasses regarding the yaw with 48.49° and average with 28.15°, pointing towards a less effective pose estimation accuracy in extreme poses when all glasses are combined. In general, this method performs better when trained on individual glasses.

GlassPose shows a similar pattern regarding errors on individual objects. We observe an average MAE between 0.50° and 0.56° and an average RMSE between 0.79° and 0.92° for the individual glasses. The pattern is similar for the individual axes. The BMAE shows higher differences: The smallest glasses model NORTH has an average of 2.84°, while the MAV results in 1.70°. One outlier is the roll BMAE for EVS, which is 4.21°, compared to values around 2.5° for all other glasses. ALL again performs worse than the individual HMD models.

The same is observable for GlassPoseRN on the individual objects. The MAE and RMSE values variate in a limited range. The pattern is similar for the individual axes. The BMAE shows fewer differences for GlassPoseRN. It ranges from 0.15° to 0.26°, showing improved estimation performance for extreme poses, while the error is still higher than the MAE and RMSE. In addition, GlassPoseRN generalized well when trained on all HMD models combined: the error values on ALL are similar to the individual glasses.

CapsPose reduces the differences between the glasses and the axes even further. The MAE and RMSE are mainly around 0.10° and 0.2°. In some cases, BMAE is even lower than the other metrics, showing the superior estimation performance of CapsPose, even for less common cases. There is no difference in estimation performance visible when trained on ALL.

Method Comparison on Orientation Results There is a significant estimation performance observable among RandomizedBins and our methods. While RandomizedBins mostly achieves errors around 10° and higher, it also performs significantly worse on the BMAE, showing an inferior performance on extreme poses. The performance is also reduced on ALL.

GlassPose presents an improved estimation performance compared to RandomizedBins. The MAE and RMSE are mostly lower than 1.00°. We still observe that performance on extreme poses is worse, as the BMAE is 3-4 times higher than the RMSE and MAE. There is also a performance reduction regarding ALL.

In contrast, GlassPoseRN achieves errors mostly below 0.2° on average. The BMAE error range is mainly similar to the other metrics, showing improved estimation performance for difficult cases. Compared to the other two methods, it also shows no difference when trained and tested on ALL.

CapsPose also performs well regarding the BMAE among all HMD models, and it also generalizes well, which the values of ALL show. CapsPose mostly improves the estimation performance in comparison the GlassPoseRN even further.

Position Results Table 7.7 shows the position results of the four approaches, trained and tested on the 94/3/3 data split for all individual glass models and combined.

Glasses-	RandomizedBins[81]				GlassPose				GlassPoseRN				CapsPose			
Type	x	y	z	L_2	x	y	z	L_2	x	y	z	L_2	x	y	z	L_2
EVS	794.96	483.75	27.48	932.43	2.35	2.14	2.21	4.50	0.59	0.53	0.39	1.02	**0.26**	**0.27**	**0.09**	**0.44**
MAV	754.36	510.57	48.21	914.04	2.11	2.00	1.95	4.03	0.55	0.49	0.43	0.99	**0.22**	**0.25**	**0.10**	**0.40**
HOLO	740.35	415.94	51.87	852.51	2.28	2.39	2.43	4.71	0.62	0.50	0.39	1.03	**0.27**	**0.30**	**0.12**	**0.47**
NORTH	837.18	595.99	40.53	1029.78	2.29	2.25	2.17	4.52	0.67	0.53	0.56	1.19	**0.29**	**0.27**	**0.13**	**0.47**
ALL	748.66	473.33	41.27	887.97	2.85	2.73	2.54	5.43	0.49	0.50	0.35	0.90	**0.25**	**0.26**	**0.13**	**0.44**

Table 7.7: Results for the positional, Euclidean error of GlassPose, GlassPoseRN, and CapsPose compared to RandomizedBins in millimeters. The Everysight Raptor is referenced as EVS, Hololens 1 as HOLO, North Focal Generation 1 as NORTH and the Mini Augmented Vision glasses as MAV. ALL stands for all glasses combined. The lowest values per glasses model are highlighted.

The extension of RandomizedBins with translation estimation does generally result in high errors. The errors are in the decimeter range on the x and y axes and the overall L_2 error. Only on the z-axis, the results stay in the centimeter range. The pattern of NORTH performing worst among all glasses continues for the position. HOLO performs best again with an L_2 error of 852.51mm.

GlassPose achieves significantly better estimation errors, mostly below 5mm. There is no significant performance difference between the individual glasses models. Similar to its orientation estimation, when trained on ALL, the position estimation is reduced, resulting in an L_2 error of 5.43mm, while the individual glasses result in errors around 4.5mm.

GlassPoseRN reduces the errors even further, showing errors around 0.5mm on the individual axes, and around 1mm for the L_2 error. Similar to the orientation, there is no significant difference between the errors on ALL and on the individual HMD models.

In contrast to the orientation estimation, CapsPose shows significant improvements regarding position estimation compared to GlassPoseRN, while the estimation errors are reduced mainly by more than half. The L_2 error is reduced to below 0.5mm, while the individual axes are even lower.

Discussion RandomizedBins trained on the HMDPose dataset shows acceptable results for orientation. Our extension of the method with the goal of position estimation shows poor performance. This may stem from the position not being on a continuous scale like the orientation, making the classes for RvC hard to design in an overall fitting way. This is easier for orientation, as the approach trains on Euler angles, consisting of a finite space. Additionally, the inputs of RandomizedBins in the original work are RGB images instead of IR images. The original work also focuses on head pose estimation in contrast to object pose estimation, which we conducted in this work.

GlassPose reduced all errors significantly but has similar problems like RandomizedBins: it does not generalize well when all glasses models are combined, and the extreme poses are estimated worse. Still, the error is reduced to mostly less than 1° for orientation and 5mm for position.

GlassPoseRN, in contrast, does not have a problem with extreme poses. It also shows a good generalization performance on ALL, as no difference is visible compared to the individual glasses. The error is reduced to below 1mm in position and below 0.2° in orientation.

CapsPose shows similar qualities to GlassPoseRN, while simultaneously reducing the overall estimation errors further. The position error is reduced to below 0.5mm, while the orientation error is mostly around 0.1°.

7.3.2 Recurrent Pose Tracking Results

The second block we identified for a potential accurate pose estimation based on IR images was image-based recurrent pose tracking. We trained the methods presented in Section 4.2 for evaluation (Figure 7.4).

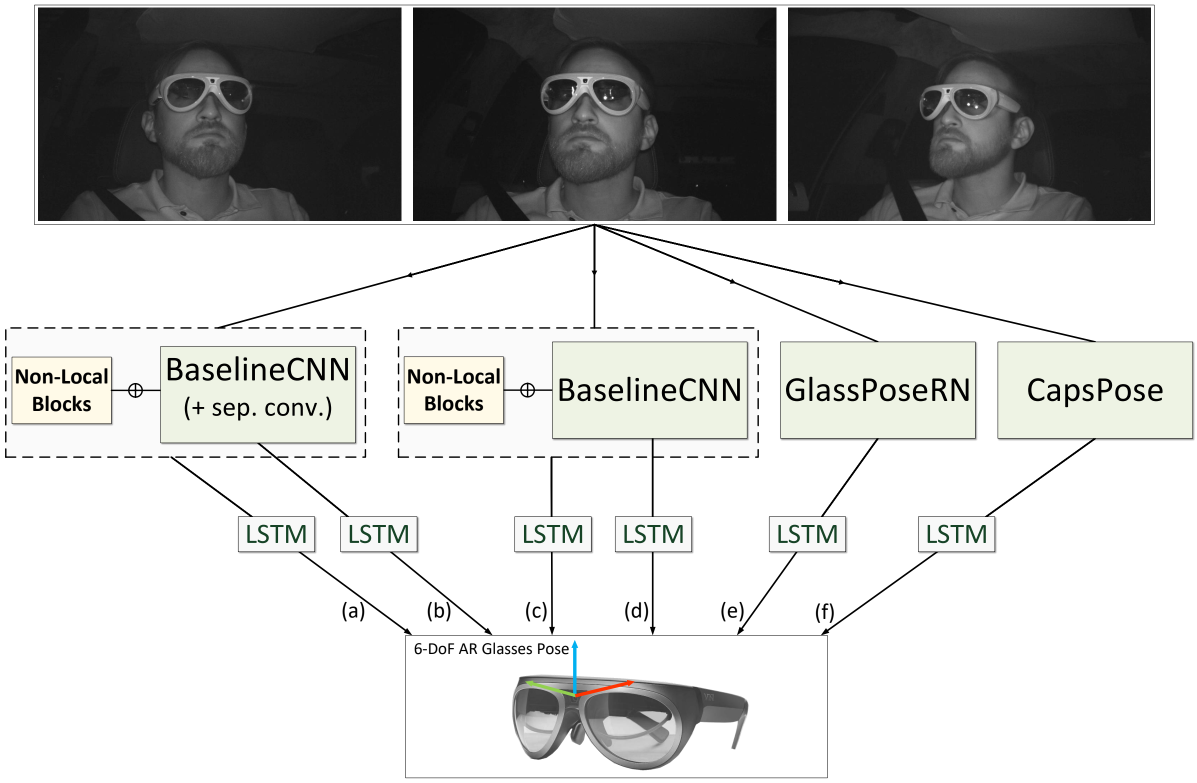


Figure 7.4: An overview of our RNN evaluation. We compare six different pose estimation and tracking approaches on triple images: In (d) and (e), we introduce LSTMs to two frame-by-frame approaches. We extend the CapsPose network with LSTMs in (f). (c) depicts the standard CNN with non-local blocks and an LSTM. (b) shows a CNN variant with separable convolutions and an LSTM, while (a) extends the approach with non-local blocks.

Dataset Preparation for Recurrent Learning

The dataset preparation starts again with a downscaling of the 1280 × 752 pixel images to 320 × 188 with a following normalization step. Next, the three images from different views from the same timestep are stacked together channel-wise. We create sequences from these multi-view images using the individual frames' timestamp as required by RNN-based networks. The data split is acquired while sequence generation is used for all the networks. Figure 7.5 gives an overview of the pipeline of preprocessing such sequences.

Our training, validation, and test split is 80% for training and 10% for validation and test set. In contrast to the 94/3/3 split used in the previous section for the image-based feed-forward methods, more extensive test and validation sets are required, as sequencing data comes with the loss of frames. We split the dataset for all individual glasses types as well as for individual subjects.

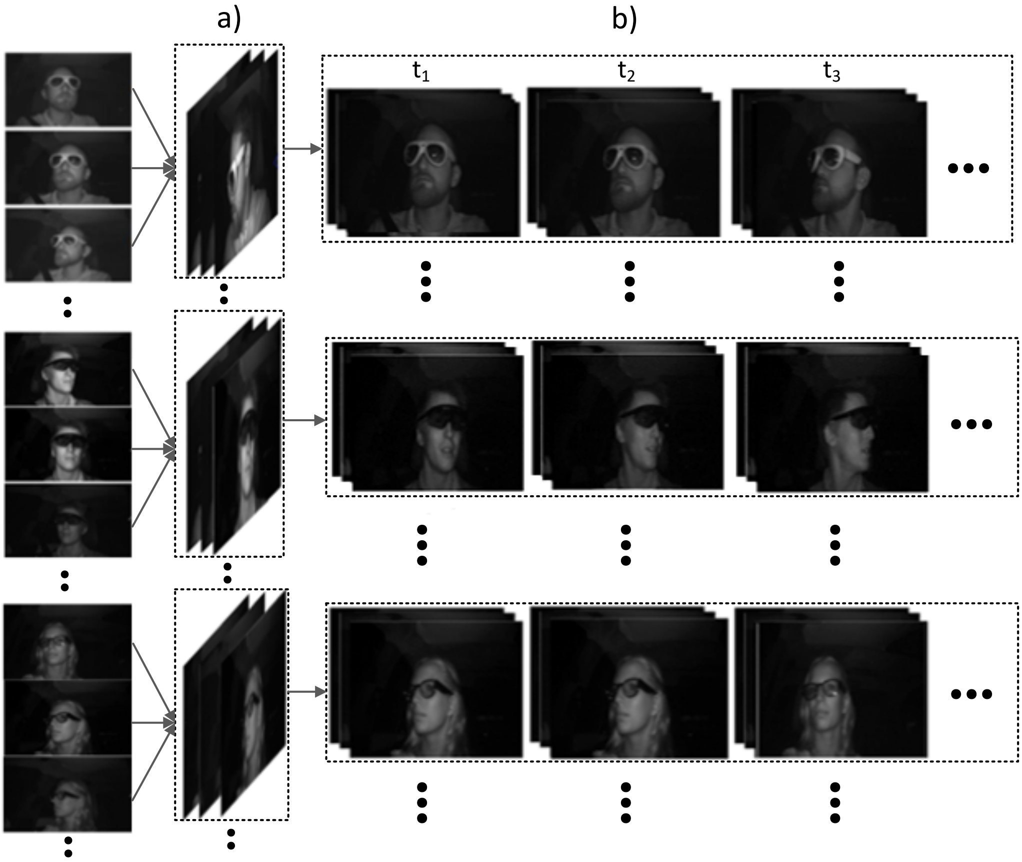


Figure 7.5: Our pipeline to conduct sequence generation from multi-view IR images for LSTM-based methods. (a) We first stack IR images channel-wise and then (b) generate sequences of the stacked images according to their timestamp.

Evaluation Metrics for Recurrent Neural Networks

We consider only the MAE and RMSE for orientation. An additional metric is the BMAE, which takes the unbalanced amount of the full range of head orientations by introducing sections into account. Despite its interesting insight into a networks' performance over the complete range, we exclude this metric, as the section definition is invalid for sequenced data. This would implicate the definition of ranges per sequence, where the sample size is too small to represent the test set. We utilize the L_2 loss for the position error on all axes separately and together for the position estimation.

LSTM-based Pose Tracking Methods

Based on the BaselinCNN, GlassPoseRN, and CapsPose methods, we introduced six different pose tracking approaches in Section 4.2. We evaluate them on individual glasses models and their combinations. Table 7.8 shows the orientation results.

Glasses-		CNN-LSTM				SepConv-LSTM				NL-CNN-LSTM				NL-SepConv-LSTM				GlassPoseRN-LSTM				SeqCapsPose			
Type	Metric	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg
EVS	MAE	0.65	0.63	0.94	0.74	0.64	0.63	1.01	0.76	0.62	0.66	1.01	0.76	0.99	1.01	1.81	1.27	0.99	1.74	1.31	1.34	**0.30**	**0.40**	**0.77**	**0.49**
	RMSE	0.67	**0.65**	**1.00**	**0.77**	0.66	**0.65**	1.07	0.79	**0.65**	0.68	1.06	0.80	1.04	1.05	1.93	1.34	1.06	1.79	1.43	1.43	0.87	1.36	4.44	2.23
MAV	MAE	0.84	0.83	0.77	0.81	0.86	0.73	0.76	0.79	0.82	0.88	0.76	0.82	0.88	0.72	0.74	0.78	1.30	1.10	1.25	1.22	**0.41**	**0.46**	**0.47**	**0.45**
	RMSE	0.86	0.85	0.82	0.84	0.89	**0.75**	0.81	**0.82**	**0.84**	0.90	0.81	0.85	0.91	**0.75**	**0.80**	**0.82**	1.42	1.22	1.47	1.37	0.84	1.30	1.40	1.18
HOLO	MAE	0.85	0.76	1.21	0.94	0.84	0.74	**1.00**	0.86	0.96	0.80	1.16	0.97	2.68	1.62	3.41	2.57	1.4	2.14	1.47	1.67	**0.57**	**0.49**	1.07	**0.71**
	RMSE	**0.87**	0.78	1.26	0.97	**0.87**	**0.76**	**1.06**	**0.90**	0.98	0.83	1.23	1.01	2.70	1.81	3.53	2.68	1.47	2.20	1.65	1.77	1.37	1.39	4.98	2.58
NORTH	MAE	0.76	0.67	1.22	0.89	0.90	0.75	1.09	0.91	0.84	0.74	1.10	0.89	1.63	2.19	3.15	2.33	0.91	1.67	1.52	1.37	**0.51**	**0.62**	**0.75**	**0.62**
	RMSE	**0.79**	**0.72**	1.30	**0.94**	0.93	0.80	1.18	0.97	0.86	0.80	**1.17**	**0.94**	1.69	2.28	3.35	2.44	0.97	1.77	1.67	1.47	2.33	2.19	4.61	3.04
ALL	MAE	0.81	0.84	0.98	0.88	0.74	0.72	0.98	0.81	0.91	0.87	1.06	0.94	0.84	0.83	1.11	0.92	1.30	3.13	1.25	1.90	**0.23**	**0.35**	**0.34**	**0.31**
	RMSE	0.83	0.87	1.05	0.92	0.77	**0.77**	1.05	0.86	0.95	0.91	1.14	1.00	0.87	0.88	1.18	0.98	1.36	3.19	1.39	1.98	**0.44**	0.80	**0.71**	**0.65**

Table 7.8: Rotation results our LSTM-based Pose Trackers CNN-LSTM, SepConv-LSTM, NL-CNN-LSTM, NL-SepConv-LSTM, GlassPoseRN-LSTM, and SeqCapsPose on the given error metrics in degrees. The Everysight Raptor is referenced as EVS, Hololens 1 as HOLO, North Focal Generation 1 as NORTH and the Mini Augmented Vision glasses as MAV. ALL stands for all glasses combined. The roll, pitch, yaw and the average of all three axes are given on the defined metrics. The lowest values per glasses model are highlighted.

GlassPoseRN-LSTM estimates the orientation with less than 1.98° error. Regarding the individual glasses, we observe a similar pattern as for GlassPoseRN. ALL has a lower performance than the glasses individually. On NL-SepConv-LSTM, the error is high for NORTH and HOLO, being the smallest and the largest glasses in the dataset. The average error of all other glasses models and their combination is below 1.50°. The performance of CNN-LSTM, SepConv-LSTM, and NL-CNN-LSTM is comparable when differences on individual glasses are considered. The averages of all three methods for MAE on the combined glasses models range from 0.74° to 0.94°. On the RMSE, the average values range from 0.77° to 1.01°. Regarding all five methods in comparison, we observe the overall consistently best performance on the SepConv-LSTM. The method achieves the best results on ALL and HOLO on both metrics. The only network achieving better results is SeqCapsPose. It performs differently on individual glasses compared to SepConv-LSTM. The errors on HOLO and NORTH glasses are relatively higher than on the other glasses. This may be explained by the small size of the NORTH glasses and the complex shape of the HOLO glasses. In general, SeqCapsPose demonstrates consistently lower errors on the MAE. However, the RMSE of SeqCapsPose is higher than SepConv-LSTM.

The position benchmark results in a comparable pattern (Table 7.9). In the case of GlassPoseRN-LSTM, the errors are the highest, with an L_2 error of 15.86mm on ALL and values between 5.65mm to 11.83mm on the individual glasses. NL-SepConv-LSTM has higher errors on HOLO and NORTH, as already seen on the orientation error. NL-CNN-LSTM, Sep-Conv-LSTM, and CNN-LSTM attain errors in a similar range for the individual glasses and combined. Although close error ranges between the three aforementioned approaches are observable, SepConv-LSTM performs consistently best. Similar to our orientation comparison, NL-SepConv-LSTM

Glasses-	CNN-LSTM				SepConv-LSTM				NL-CNN-LSTM				NL-SepConv-LSTM				GlassPoseRN-LSTM				SeqCapsPose			
Type	x	y	z	L_2	x	y	z	L_2	x	y	z	L_2	x	y	z	L_2	x	y	z	L_2	x	y	z	L_2
EVS	2.86	1.74	1.44	4.18	2.84	1.83	1.06	3.96	2.90	2.19	1.64	4.55	4.73	3.70	2.11	7.18	6.43	2.74	3.52	8.89	**1.78**	**1.01**	**0.57**	**2.36**
MAV	2.85	1.66	1.62	4.23	2.19	1.27	1.21	3.22	2.29	1.59	1.37	3.59	2.55	1.40	1.18	3.56	4.11	2.02	1.99	5.65	**1.50**	**0.83**	**0.46**	**2.01**
HOLO	3.38	2.51	1.89	5.31	3.15	2.02	1.42	4.57	3.52	2.94	2.36	5.93	6.80	3.20	5.65	11.67	5.39	3.41	4.79	9.39	**2.12**	**1.99**	**0.96**	**3.43**
NORTH	3.13	2.70	1.73	5.18	3.02	2.52	1.70	5.09	3.24	2.70	1.77	5.19	7.45	6.50	5.27	12.79	6.38	3.89	6.55	11.83	**1.90**	**1.59**	**0.76**	**3.00**
ALL	3.01	2.70	1.97	5.22	2.67	2.43	1.53	4.46	3.48	2.67	1.85	5.44	3.40	2.69	1.91	5.43	8.55	7.67	7.52	15.86	**0.97**	**0.87**	**0.50**	**1.60**

Table 7.9: Results for the positional, Euclidean error in millimeters of our LSTM-based Pose Trackers CNN-LSTM, SepConv-LSTM, NL-CNN-LSTM, NL-SepConv-LSTM, GlassPoseRN-LSTM, and SeqCapsPose on the individual axes and combined. The Everysight Raptor is referenced as EVS, Hololens 1 as HOLO, North Focal Generation 1 as NORTH and the Mini Augmented Vision glasses as MAV. ALL stands for all glasses combined. The lowest values per glasses model are highlighted. The lowest values per metric are highlighted.

and GlassPoseRN-LSTM achieve higher errors than the rest. The only exception is again SeqCapsPose, outperforming all other networks consistently on all glasses models and axes.

AR Glasses Pose Estimation & Tracking Comparison

	Frame-by-Frame Pose Estimation			LSTM-based Pose Tracking					
Metric	Baseline -CNN	GlassPose -RN	CapsPose	CNN -LSTM	SepConv -LSTM	NL-CNN -LSTM	NL-SepConv -LSTM	GlassPoseRN -LSTM	Seq -CapsPose
MAE	2.98	3.86	**0.27**	0.88	0.81	0.94	0.92	1.90	0.31
RMSE	4.33	5.26	**0.65**	0.92	0.86	1.00	0.98	1.98	**0.65**

Table 7.10: Rotation results of the Frame-by-Frame Pose Estimators BaselineCNN, GlassPoseRN, and CapsPose as well as our LSTM-based Pose Trackers CNN-LSTM, SepConv-LSTM, NL-CNN-LSTM, NL-SepConv-LSTM, GlassPoseRN-LSTM, and SeqCapsPose on the given error metrics in degrees. The average value of all three axes is given on the defined metrics in this table.

Our fundamental goal in this section is to compare pose estimation and pose tracking methods for AR glasses. For this purpose, we benchmarked three frame-by-frame pose estimation and six pose tracking approaches. The data split 80/10/10 is used again for the feed-forward CNN methods, but the frames are being shuffled. The approaches mostly show significant improvements in favor of pose tracking methods, except for CapsPose.

	Frame-by-Frame Pose Estimation			LSTM-based Pose Tracking					
Metric	Baseline -CNN	GlassPose -RN	CapsPose	CNN -LSTM	SepConv -LSTM	NL-CNN -LSTM	NL-SepConv -LSTM	GlassPoseRN -LSTM	Seq -CapsPose
L_2	14.82	11.95	**1.43**	5.22	4.46	5.44	5.43	15.86	1.60

Table 7.11: Results for the positional, Euclidean error in millimeters of the Frame-by-Frame Pose Estimators BaselineCNN, GlassPoseRN, and CapsPose as well as our LSTM-based Pose Trackers CNN-LSTM, SepConv-LSTM, NL-CNN-LSTM, NL-SepConv-LSTM, GlassPoseRN-LSTM, and SeqCapsPose on all three axes combined. The lowest value is highlighted.

Pose Accuracy Regarding orientation results, we generally observe a dominance of Capsule Network-based approaches (Figure 7.10). Our extension of the GlassPoseRN method with LSTM performs better than GlassPoseRN. Regarding pose accuracy, the second-best frame-by-frame method GlassPoseRN achieves orientation errors on the MAE and RMSE of 3.86° and 5.26°, respectively. This is drastically reduced by CapsPose, achieving 0.27° and 0.65° for the MAE and RMSE. In terms of recurrent pose estimators, SepConv-LSTM achieves low errors as small as 0.81° and 0.86°, which is considerably lower than the BaselineCNN and GlassPoseRN methods. Still, SeqCapsPose achieves lower results than SepConv-LSTM with a 0.31° MAE and 0.65° RMSE. The values are very similar to the CapsPose results. An analogous tendency is visible on the position estimation (Figure 7.11). Despite SepConv-LSTM achieving an L_2 error of 4.46mm and being significantly lower than GlassPoseRN with 11.95mm, the Capsule Network-based algorithms decrease the error further. The frame-by-frame method CapsPose results in 1.43mm, while SeqCapsPose achieves 1.60mm.

Inference Time Comparing their performance regarding inference time is possible by measuring the estimation time between feeding images into a network, and the time a network predicts. Our measurements were made on a single NVIDIA GeForce 2080Ti. The BaselineCNN achieves 100FPS, compared to 42FPS of GlassPoseRN and 20FPS of CapsPose. The pose tracking methods except for SeqCapsPose all achieve similar inference times due to their similarity in depth. Thus, each method achieves 74FPS. SeqCapsPose stays at an inference time of 20FPS.

Discussion

We chose our methods with a mostly identical CNN-backbone to ensure a network as similar as possible. This had the aim to deduce if the LSTM, the additional non-local blocks, or separable convolution deployment contributes to the changes in the accuracy. Between the numerous LSTM-based methods combined with the BaselineCNN, the combination of non-local blocks and separable convolutions decreased estimation accuracy compared to approaches with the individual components separately. An explanation for this might be the concurring properties of separable convolutions and non-local blocks. Separable convolutions separately handle distinct channels of the images by applying the convolutions individually, which are the individual IR images in our case. Non-local blocks aim to learn non-local dependencies in the images. The layers containing non-local blocks might enjoy the information of three mixed images, which collides with the separable convolution concept. Individually applied, they output similar errors.

Finally, the introduction of LSTMs to the ResNet-18-based GlassPoseRN brings improvement but underperforms compared to the methods based on a simple CNN. This underlines once more the enhancement of the pose estimation accuracy with the introduction of LSTMs. The advanced feature extraction properties of ResNet seem redundant in the case of IR image-based AR glasses pose estimation when LSTMs are deployed. In addition, our combined methods work more efficiently compared to the GlassPoseRN approach. We achieve around 74FPS inference time for the BaselineCNN-LSTM combinations, whereas the inference time for GlassPoseRN is 42FPS.

One outlier to the observations above is CapsPose and SeqCapsPose. Both perform best among their counterparts and achieve similar error levels. The introduction of LSTMs does not seem to enhance the estimation quality if a Capsule Network backbone is used, as the CapsPose network achieves a very low error already. The temporal variation of head features might not have been well understood by Capsules. Therefore, the idea of introducing recurrent blocks to Capsule Networks is an open research problem.

7.3.3 Point Cloud-based Pose Estimation Results

Finally, we present the last part of our fundamental IR image-based approaches to estimate 6-DoF AR glasses pose estimation: derivation of point clouds based on infrared images for further pose estimation. This first requires the generation of point clouds.

Point Cloud Generator

We presented the point cloud generator Subsection 4.3.1. The networks were trained on the initial data split, where our training, validation, and test split is 94% for training and 3% for validation and test set each. Before splitting, the dataset is being shuffled. Based on the trained network, we generated point clouds for all images triples of the dataset, including the images contained in the training, validation, and test set.

Results

We used the point clouds to train our networks P2R and P2P and the state-of-the-art CloudPose method. The same 94/3/3 data split was used for training, as done for the image-based feed-forward pose estimation networks. After training CloudPose [92], P2R, and P2P methods for 30 epochs each, we evaluate the translation and orientation error on the defined metrics.

Orientation Results Table 7.12 shows the error of the three methods on the defined metrics for the orientation on all axes individually and on average.

Glasses-		CloudPose[92]				P2R				P2P			
Type	Metric	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg
EVS	MAE	0.93	0.81	1.18	0.97	**0.41**	**0.37**	**0.53**	**0.43**	0.46	0.38	**0.53**	0.46
	RMSE	1.46	1.19	1.52	1.39	**0.60**	**0.53**	0.87	0.66	0.64	**0.53**	**0.79**	**0.65**
	BMAE	6.52	0.91	1.69	3.04	**0.74**	**0.48**	2.10	1.11	1.28	0.64	**0.79**	**0.90**
MAV	MAE	0.90	0.65	1.34	0.96	**0.48**	**0.38**	**0.52**	**0.46**	**0.48**	0.39	0.54	0.47
	RMSE	1.21	0.87	1.99	1.36	0.71	**0.53**	**0.81**	0.69	**0.66**	0.54	**0.81**	**0.67**
	BMAE	2.28	0.82	3.04	2.05	**0.83**	**0.51**	0.79	**0.71**	0.91	0.63	**0.69**	0.74
HOLO	MAE	0.88	0.77	1.03	0.90	**0.43**	**0.34**	**0.49**	**0.42**	0.47	0.37	0.53	0.46
	RMSE	1.20	1.10	1.44	1.25	**0.61**	**0.50**	**0.75**	**0.62**	0.65	0.52	0.77	0.65
	BMAE	2.12	0.83	1.48	1.48	**0.61**	**0.40**	1.12	**0.71**	0.90	0.44	**0.82**	0.72
NORTH	MAE	1.05	0.78	1.52	1.12	**0.50**	**0.55**	**0.59**	**0.55**	0.55	0.57	0.63	0.58
	RMSE	1.41	1.11	1.83	1.45	0.78	**0.86**	0.95	0.87	**0.76**	0.87	**0.91**	**0.84**
	BMAE	2.11	1.72	2.09	1.98	2.80	**2.58**	2.04	2.47	**1.29**	2.66	**1.09**	**1.68**
ALL	MAE	1.20	0.90	0.78	0.96	**0.51**	**0.46**	**0.58**	**0.52**	0.53	0.47	0.61	0.53
	RMSE	1.48	1.13	1.12	1.24	**0.73**	0.69	**0.88**	0.77	**0.73**	**0.68**	**0.88**	**0.76**
	BMAE	2.41	0.99	1.72	1.71	**1.21**	**0.56**	1.50	**1.09**	1.81	**0.56**	**1.26**	1.21

Table 7.12: Orientation results of the P2R and P2P approaches compared to CloudPose on the given error metrics in degrees. The Everysight Raptor is referenced as EVS, Hololens 1 as HOLO, North Focal Generation 1 as NORTH and the Mini Augmented Vision glasses as MAV. ALL stands for all glasses combined. The roll, pitch, yaw and the average of all three axes are given on the defined metrics. The lowest values per glasses model are highlighted.

The errors for CloudPose are close among the individually trained and combined glasses models. THE MAE is below 1.12° on average, which is the highest for the smallest glasses model NORTH. The average RMSE is between 1.24° and 1.45°, with the lowest value for ALL and HOLO, and the highest for NORTH. In return, the BMAE is lower for NORTH with 1.98° in contrast to 3.04° for EVS. This shows that NORTH performs better in predicting extreme poses.

The errors for P2R are comparable in the case of individually trained and combined glasses. The MAE error of the P2R algorithm is lower than 0.59° on all axes for all glasses models. For HOLO, we achieve the lowest errors with an average error of 0.42° overall on the MAE. NORTH results in the highest error for MAE. The BMAE shows good estimation quality even for difficult cases like extreme poses among all objects except NORTH, the smallest model in the dataset. NORTH results in more than twice as high error on the BMAE with an average of 2.47° in contrast to all other glasses. Larger glasses models like HOLO and MAV perform better with an average BMAE of 0.71° than the smaller ones. The results for ALL are as expected, as they are in the mid-range considering all other glasses individually.

Our P2P method performs similarly to the P2R method. There is little difference observable among all objects. P2P results in average estimation accuracy on ALL compared to the objects individually. On the RMSE metric HOLO and EVS result in the lowest overall error with average errors of 0.65°. The smallest glasses model NORTH results in the highest error compared to the other object individually on all metrics. The error on ALL is in the mid-range compared to the individual objects again, with an average RMSE of 0.76°. On the BMAE metric, P2P performs best on the largest model HOLO with an average error of 0.72°, while achieving an average error of 1.68° for the smallest model NORTH. In general, P2P improves with larger AR glasses sizes.

We can generally observe significant improvements in favor of the P2R and P2P methods as opposed to CloudPose for the orientation regression. From the error levels of CloudPose to our P2R and P2P methods, we can observe an error reduction of around 50%. One example of this is the roll for BMAE on ALL, which drops from 2.41° to 1.21°. A similar reduction can be seen between the average MAE error for ALL, which is 0.96° for CloudPose and 0.51° for P2R. Generally, P2R and P2P result in comparable errors. On NORTH, MAV, EVS, and ALL, we observe close errors for almost all axes individually and on average. For example, the averages for MAV on the MAE are 0.47° for P2P compared to 0.46° for P2R. Similarly, the BMAE is 0.74° for P2P and 0.71° for P2R. One exception is BMAE for NORTH, where P2P performs better with an average of 1.68° against 2.47° for P2R. Despite the errors being close for HOLO, P2R performs better on HOLO in general on most measured errors. In conclusion, P2P and P2R achieve similar results, while P2P performs more consistent as observable in our NORTH example.

Position Results

Table 7.13 shows the positional L_2 error of the CloudPose, P2R, and P2P methods on all individual axes and in total.

CloudPose achieves position errors between 4.28mm and 7.86mm on the z-axis. Equivalently to the z-axis, the errors on the y-axis and the L_2 error have low variation between the models. The x-axis shows errors between 11.20mm for NORTH and 4.91mm for MAV, showing higher errors for smaller glasses models.

As P2R does not regress the position, we illustrate the point cloud estimator's position results. For P2R, the position error on the x- and y-axis are similar, ranging from 3.59mm to 4.14mm on the x-axis and from 2.59mm to 3.29mm on the y-axis. The error is the lowest on the z-axis, ranging from 1.38mm for HOLO to 1.54mm for MAV. The model ALL presents a similar pattern for the orientation, where the error is in the mid-range compared to the other objects individually.

For P2P, the position estimation is similar on all four objects. Regarding the individual axes, we observe errors in similar ranges. The values for the individual axes are between 2.17mm and 3.50mm. The overall L_2 distance starts from 5.07mm for EVS and ranges to 6.15mm for NORTH. The outcome for ALL is in the mid-range compared to all individual glasses separately.

Glasses-	CloudPose[92]				P2R*				P2P			
Type	x	y	z	L_2	x	y	z	L_2	x	y	z	L_2
EVS	10.90	8.06	5.65	15.86	3.68	3.11	**1.40**	5.66	**2.95**	**2.23**	2.64	**5.26**
MAV	4.91	7.29	4.28	11.06	3.80	2.68	**1.54**	5.49	**2.86**	**2.17**	2.55	**5.07**
HOLO	7.87	7.43	7.86	14.81	3.59	3.29	**1.38**	5.71	**3.03**	**2.32**	2.65	**5.35**
NORTH	11.20	6.09	5.92	15.85	4.14	2.59	**1.40**	**5.66**	**3.50**	**2.51**	3.15	6.15
ALL	7.83	7.87	7.76	15.28	3.85	2.92	**1.40**	**5.68**	**3.24**	**2.38**	2.96	5.75

* Since P2R doesn't regress the position, we show the position results taken from the point cloud estimator.

Table 7.13: Results for the positional, Euclidean error of P2P and P2R compared to CloudPose in millimeters. The Everysight Raptor is referenced as EVS, Hololens 1 as HOLO, North Focal Generation 1 as NORTH and the Mini Augmented Vision glasses as MAV. ALL stands for all glasses combined. The lowest values per glasses model are highlighted.

P2P mostly delivers the best position estimation. For the x and y axes and the L_2 distance, P2P brings the most notable performance improvement. On the contrary, P2R attains the best outcomes consistently on the z-axis, where the errors are between 1.66mm and 2.24mm.

Discussion

For the orientation, we observe improvement overall from CloudPose to our point cloud-based methods. We can also see better estimation results with point cloud-based methods for extreme poses, which the BMAE metric shows consistently. Compared to the CloudPose method, our approaches bring improvements in orientation and position estimation. Another observation for the individual objects is the lower error for larger objects. HOLO results in the overall lowest errors for all methods, whereas NORTH results in the highest error. Between the P2R and the P2P estimation, minor differences are observable. Still, the P2P method estimates a slightly better orientation, also for extreme poses. This is visible on the BMAE metric. For position regression, P2P performs best among all methods. The values on the y-axis and the L_2 error on all axes at once are lower most of the time. One exception is the z-axis, where P2R performs better.

Furthermore, P2R shows mostly close results to P2P, making it a good option for efficient deployment due to less computation requirement. P2P is preferable in case of better resource availability to improve estimation accuracy.

7.4 Head and AR Glasses Pose Estimation Comparison

In previous sections of this chapter, we evaluated various HMD pose estimation architectures. They were either solely image-based feed-forward, image-based recurrent, or point cloud-based pose estimation and tracking techniques. As presented in Section 5.2, we deployed a selection of them on head pose estimation. For a fair and in-depth analysis, we trained the different methods on several data splits. The goal is to compare their performances if the glasses are known, but the face is unknown, the glasses are unknown, but the face is known, or both are known to the networks during training.

7.4.1 Data Split Strategies

The HMD and head annotations of the HMDPose dataset are acquired from several subjects wearing different glasses yielding a large inter-class variation. Therefore, we define different types of data split strategies (Figure 7.6).

IntraALL-RND shuffles all acquired frames and randomly selects portions of 94%, 3% and 3% for training, validation and testing sets. The evaluation using this split shows us how good a Neural Network predicts poses from the infrared input images. In that strategy, all types of glasses and all subjects are present in all sets.

InterGlasses-RND uses three types of glasses for the training set and one type of glasses for the validation and testing sets. This split is interesting to test the ability of Networks in generalizing glasses. This type of evaluation is needed when a new type of glasses is used, on which the pose estimator was not trained.

InterSubjects-RND split technique is similar to InterGlasses-RND, where the split is according to the class of subjects. Only 10 subjects are included in the training set and two other different subjects for each of the validation and test sets. The evaluation results show the performance of the Networks in generalizing the subjects, and thus, its performance on an arbitrary person.

IntraALL-SEQ split strategy utilized for recurrent models. The training, validation and test sets respectively include 80%, 10% and 10% of each sequence. In that strategy, all types of glasses and all subjects are present in all sets, as in IntraALL-RND.

7.4.2 Head and HMD Comparison Results

Following our data split strategies, we compare the head pose and HMD pose estimators trained on them. This subsection contains the results on feed-forward pose estimators and recurrent pose trackers.

Feed-Forward Pose Estimation

To compare the head and glasses pose estimation performance, we first investigate the estimation accuracy of feed-forward networks trained on the HMDPose dataset with HMD and head

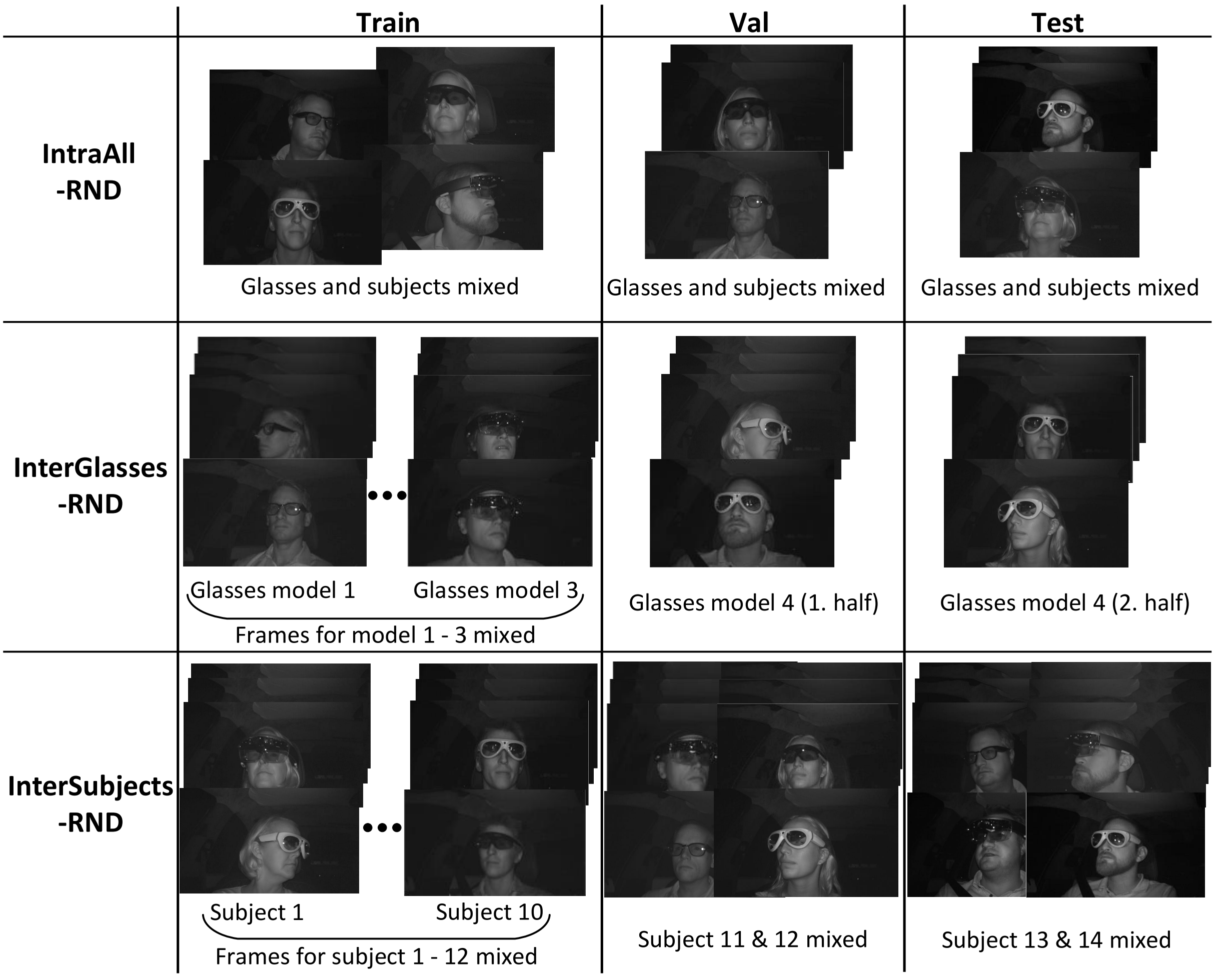

Train
Val
Test
IntraAll -RND
Glasses and subjects mixed
Glasses and subjects mixed
Glasses and subjects mixed
InterGlasses -RND
Glasses model 1
Glasses model 3
Frames for model 1 - 3 mixed
Glasses model 4 (1. half)
Glasses model 4 (2. half)
InterSubjects -RND
Subject 1
Subject 10
Frames for subject 1 - 12 mixed
Subject 11 & 12 mixed
Subject 13 & 14 mixed

Figure 7.6: An overview of the different split strategies IntraAll-RND, InterGlasses-RND and InterSubjects-RND.

pose annotations. For the sake of fairness, the same input frames are fed to the head and glasses pose estimators in the training and the evaluation phases.

Table 7.14 summarizes the orientation estimation results of the models trained on HMDPose with HMD and head annotations, for glasses and head pose estimation on the IntraALL-RND split strategy. The corresponding position estimation results are illustrated in Table 7.15.

GlassPose-HMD reduces the orientation estimation error compared to GlassPose-HEAD with a reduction of the MAE of up to 38% and the rotational RMSE of up to 30%. Also, we notice that the position error is lowered by up to 27%. There are multiple explanations for this. The pixels brightness of the glasses in an IR image is significantly higher than that of the head. This facilitates the search of the target object by the Neural Network. Furthermore, the shape of the glasses is not uniform, i.e., many edges could be extracted from the glasses, which may improve the quality of the extracted feature map. Moreover, glasses have roughly a rectangular form, and extracting the orientation of such a simple form is easy. On the contrary, the head

	MAE				RMSE				BMAE			
Model	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg
GlassPose-HMD	**0.67**	**0.60**	**0.61**	**0.63**	**1.06**	**0.98**	**1.03**	**1.02**	4.44	**1.69**	5.54	3.89
GlassPose-HEAD	0.84	0.77	1.45	1.02	1.19	1.13	2.05	1.46	**3.16**	2.62	**1.71**	**2.50**
GlassPoseRN-HMD	**0.06**	**0.10**	**0.14**	**0.10**	**0.09**	**0.25**	**0.22**	**0.19**	**0.29**	**0.12**	**0.31**	**0.24**
GlassPoseRN-HEAD	0.20	0.20	0.47	0.29	0.28	0.29	0.73	0.43	0.71	0.43	0.50	0.55
P2P-P-HMD	**0.39**	**0.43**	0.75	**0.52**	**0.55**	**0.57**	1.09	**0.74**	**0.72**	**0.55**	2.14	**1.14**
P2P-P-HEAD	0.44	0.51	1.58	0.84	0.60	0.67	2.14	1.14	1.07	1.27	**1.80**	1.38
P2P-F-HMD	0.53	0.47	**0.61**	0.53	0.73	0.68	**0.88**	0.76	1.81	0.56	**1.26**	1.21
P2P-F-HEAD	2.70	5.32	4.93	4.31	3.66	6.54	6.53	5.58	5.42	8.46	7.48	7.12
CapsPose-HMD	**0.05**	**0.09**	**0.12**	**0.09**	**0.08**	0.24	**0.21**	**0.18**	**0.09**	**0.09**	**0.21**	**0.13**
CapsPose-HEAD	0.22	0.23	0.54	0.33	0.22	**0.23**	0.54	0.33	0.35	0.28	0.50	0.38

Table 7.14: Orientation results in degrees of all feed-forward models for head and glasses pose estimation trained and evaluated on the HMD and head labels, respectively. The IntraALL-RND data split strategy is followed. The minimum errors between glasses and head pose estimators are bold.

Model	X	Y	Z	L_2
GlassPose-HMD	**2.85**	**2.73**	**2.54**	**5.43**
GlassPose-HEAD	2.87	5.50	2.55	7.53
GlassPoseRN-HMD	**0.49**	**0.50**	**0.35**	**0.90**
GlassPoseRN-HEAD	0.86	1.12	0.57	1.74
P2P-P-HMD	10.98	6.55	17.99	25.72
P2P-P-HEAD	**4.21**	**4.26**	**4.90**	**9.01**
P2P-F-HMD	**3.24**	**2.39**	**2.96**	**5.75**
P2P-F-HEAD	33.39	31.02	65.87	89.02
CapsPose-HMD	**0.25**	**0.26**	**0.13**	**0.44**
CapsPose-HEAD	0.68	0.99	0.34	1.40

Table 7.15: Position results in millimeters of all models for head and glasses pose estimation trained and evaluated on the HMD and head labels, respectively. Feed-forward models follow the IntraALL-RND data split strategy. The minimum errors between glasses and head pose estimators are bold.

could be represented as an oval-shaped ball, and estimating its pose based on 2D images is challenging. Therefore, the network relies on the head's general form and other parts such as the nose and the mouth. GlassPose-HEAD generally shows lower errors in terms of the BMAE.

Thus, GlassPose-HMD is less effective regarding HMD pose estimation than GlassPose-HEAD for head pose estimation. The lower performance of GlassPose-HMD for extreme rotations is probably related to the insufficiency of significant features of AR glasses, as most of them are not visible when the head is turned. In contrast, the head has a bigger form, and parts of it are always visible even when performing extreme rotations.
GlassPoseRN-HMD improves over GlassPoseRN-HEAD by reducing the rotational MAE of up to 65% and the rotational RMSE of up to 55%. The translational error also improves by up to 48%. The glasses' brightness can again explain the difference in the images, their distinct appearance, and the more challenging form of the head for tracking, similar to the GlassPose differences. The BMAE of GlassPoseRN-HMD is lower than that of GlassPoseRN-HEAD. Also, we observe that GlassPoseRN-HMD produces better translation estimation performance than GlassPoseRN-HEAD with an error reduction of 48%. We especially observe a high error of the GlassPoseRN-HEAD on the Y-axis.
The P2P results for head and glasses pose estimation depends on the type of point clouds fed to the network. When using partial point clouds, either including the points of the head or the glasses, P2P-P-HEAD reduces the translation error by around 65% over P2P-P-HMD. Meanwhile, the rotation performance of the head pose estimator is worse than that of the glasses pose estimator. One reason for such inconsistent results is the use of fewer points for the glasses pose estimation. These few points of the glasses are enough to result in lower rotation error. When using the whole point cloud, we see that the findings align with the results reported by the previous methods. On the one hand, we observe a low translation performance by P2P-F-HEAD, which may be related to the very similar points on the face. On the other hand, the translation error by P2P-F-HMD is reduced by 93% compared to P2P-F-HEAD, caused by the few points of the glasses being distinguishable from the rest of the cloud.
We make a similar observation evaluating CapsPose-HMD and CapsPose-HEAD. The rotation and translation performance improve by CapsPose-HMD compared to CapsPose-HEAD with an error reduction of up to 73% and 69%, respectively. The MAE and the BMAE along the yaw axis of CapsPose-HEAD are high and close, which means that this model suffers from low performance along the yaw axis for extreme poses and normal poses. The drop of performance quality is only noticeable for CapsPose-HMD in the case of extreme poses.
Generally, the glasses pose estimation methods outperform the head pose estimation methods in the case of known glasses and drivers.

When comparing glasses pose estimators trained on all types of glasses, we observe that the performance of P2P-P-HMD and P2P-F-HMD models is worse than the performance levels reported by the other models. Furthermore, the reason for the high RMSE may be caused by outliers. The high MAE error shows that predictions are not close to the ground truth, which may cause unstable and non-smooth predictions. Also, CapsPose-HMD outperforms all other methods and reduces the GlassPoseRN-HMD errors by up to 46% and 51% for rotation and translation, respectively. The BMAE along the roll axis is also considerably reduced by 68%. Since the roll range in the dataset is small, it is a notable advantage to use this model. This finding tallies with our expectations that Capsules are better in capturing variations in a relatively narrow range. The same pattern was observed for the head pose estimators when comparing the models. Overall, our CapsPose method has demonstrated a marked improvement in the quality of pose estimation.

Recurrent Pose Tracking

We conduct the same analysis for the recurrent pose trackers. Tables 7.16 and 7.17 present the orientation and position results of the best two recurrent models trained on the same IntraALL-SEQ data distribution.

	MAE				RMSE			
Model	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg
SepConv-LSTM-HMD	**0.74**	**0.72**	**0.98**	**0.81**	**0.77**	**0.77**	**1.05**	**0.86**
SepConv-LSTM-HEAD	1.70	1.84	5.39	2.98	1.73	1.90	5.48	3.04
SeqCapsPose-HMD	**0.23**	**0.35**	**0.34**	**0.31**	**0.44**	**0.80**	**0.71**	**0.65**
SeqCapsPose-HEAD	1.16	1.37	3.03	1.86	1.93	2.52	4.93	3.13

Table 7.16: Orientation results of the best two recurrent models for head and HMD tracking, trained and evaluated on the HMDPose dataset with HMD and head annotations including all types of glasses and all drivers, respectively. The IntraALL-SEQ strategy is followed to train the networks. The evaluation is conducted in the Euler space in degrees. The minimum errors between glasses and head pose estimators are highlighted.

Model	X	Y	Z	L_2
SepConv-LSTM-HMD	**2.67**	**2.43**	**1.53**	**4.46**
SepConv-LSTM-HEAD	5.67	10.15	2.84	13.37
SeqCapsPose-HMD	**0.97**	**0.87**	**0.50**	**1.60**
SeqCapsPose-HEAD	6.55	5.48	11.72	16.13

Table 7.17: Position results of the best two recurrent models for head and HMD tracking, trained and evaluated on the HMDPose dataset with HMD and head annotations including all types of glasses and all drivers, respectively. The IntraALL-SEQ strategy is followed to train the networks. The evaluation is conducted in the Euclidean space and the results are in millimeters. The minimum errors between glasses and head pose estimators are highlighted.

All results on recurrent head tracking are inferior to their AR glasses trained counterparts. Especially along the yaw axis, there exists a high prediction error. The rotation performance of SepConv-LSTM drops significantly by a factor of more than 3 for MAE and RMSE. The performance drop by a factor of 6 is even higher for SeqCapsPose from AR glasses tracking to head tracking. The translation performance for both behaves similarly. The difference of the RMSE is slightly higher for both methods when trained on the head. This can be explained by the difficulty of understanding facial features in a temporal context.

Regarding HMD tracking, SeqCapsPose-HMD performs consistently better than SepConv-LSTM-HMD. The performance difference is observable on the MAE metric, the RMSE results are closer. The models trained on the head pose also reveal some differences. SepConv-LSTM-HEAD slightly outperforms SeqCapsPose-HEAD for the orientation and position estimation.

7.4.3 Generalization Analysis

Moreover, we evaluate a network using the splits on samples of either known drivers or known glasses. Consequently, we utilize other data split strategies such as InterSubjects-RND and InterGlasses-RND to investigate the driver's and the glasses' generalization performance.

Face Generalisation Results

We use the InterSubjects-RND data split strategy to inspect the driver generalization performance, as it includes two unseen subjects in the validation set and two persons in the test set. This gives us an insight into the network performances in estimating the pose of the glasses and of the head of an unknown person. Furthermore, we analyze the glasses generalization performance. Tables 7.18 and 7.19 list the results of CapsPose trained for head and glasses pose estimation using InterSubjects-RND and InterGlasses-RND data split strategies.

	InterSubjects-RND											
	MAE				RMSE				BMAE			
Model	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg
CapsPose-HMD	**1.76**	**2.70**	**3.38**	**2.61**	**1.76**	**2.70**	**3.38**	**2.61**	**2.72**	**2.85**	**4.16**	**3.24**
CapsPose-HEAD	2.69	8.93	6.41	6.01	3.42	9.94	8.88	7.41	3.22	7.85	8.53	6.53

	InterGlasses-RND											
	MAE				RMSE				BMAE			
Model	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg	Roll	Pitch	Yaw	Avg
CapsPose-HMD	**1.60**	4.66	**3.41**	3.22	**2.11**	5.28	**4.02**	**3.80**	3.43	6.53	**3.77**	4.58
CapsPose-HEAD	2.14	**3.02**	3.67	**2.94**	2.75	**4.21**	4.82	3.93	**2.59**	**4.17**	4.36	**3.70**

Table 7.18: Orientation results of CapsPose model for head and glasses pose estimation trained and evaluated on the HMDPose and HeadPose datasets, respectively, following the InterSubjects-RND and InterGlasses-RND data split strategies. The evaluation results are conducted in the Euler space and expressed in degrees. The minimum errors between glasses and head pose estimators are highlighted.

	InterSubjects-RND				InterGlasses-RND			
Model	X	Y	Z	L_2	X	Y	Z	L_2
CapsPose-HMD	10.93	9.14	**3.69**	**16.88**	15.03	37.54	8.59	42.75
CapsPose-HEAD	**3.22**	**7.85**	8.53	16.94	**7.16**	**5.33**	**3.56**	**10.89**

Table 7.19: Position results of CapsPose for head and glasses pose estimation trained and evaluated on the HMDPose and HeadPose datasets, respectively, following The InterSubjects-RND and InterGlasses-RND data split strategies. The evaluation results are conducted in the Euclidean space and expressed in millimeters. The minimum errors between glasses and head pose estimators are highlighted.

On the InterSubjects-RND split, CapsPose-HMD outperforms CapsPose-HEAD for the orientation estimation with an error reduction of 56%. Although the glasses are known, the position estimation performance of CapsPose-HMD is still low. This indicates that the network internally relies on some head features for the glasses position estimation. Also, CapsPose-HEAD shows high errors as the network regresses the pose from unseen heads. CapsPose-HEAD predictions seem to be unstable, as shown by the RMSE. Relatively high errors are also observed for extreme poses compared to CapsPose-HMD. This is caused by the network strongly relying on head features for the head pose estimation. CapsPose-HMD and CapsPose-HEAD result in comparable performance for head and glasses translation regression i.e. the networks make use of a similar pool of features mainly consisting of head features due to the observed high errors.

AR Glasses Generalisation Results

Concerning the InterGlasses-RND split, CapsPose-HMD and CapsPose-HEAD show the difference in the orientation performance. CapsPose-HMD produces less accurate results since the network uses some unseen features of the glasses to predict the pose. CapsPose-HEAD is also showing relatively high errors, although the driver's head is known. This indicates that the head pose estimator relies on the head features and some glasses features. However, CapsPose-HMD strongly uses glasses features, and some head features yielding acceptable rotation results. As the glasses are unknown, the RMSE demonstrates that the predictions are unstable, especially for extreme poses as proven by the high BMAE. In sum, CapsPose-HEAD is more unstable in general and more stable in extreme poses than CapsPose-HMD. The translation errors drop for head pose estimation by 74%.

AR glasses generalization is generally more useful than face generalization for real-world conditions. Existing networks can be trained with transfer learning for new glasses models, while a training for new drivers is more difficult. Thus, the overall positive glasses pose performance is beneficial.

7.5 Evaluation of Fusion Methods

In this section, we present how well each fusion algorithm performed with different ensemble constellations and which ensembles led to improved predictions. We first describe our selection of model combinations of various ensembles and how we combine them.

7.5.1 Ensemble Selection

Before evaluating the results, we define which models we use for the combinations. For this, two types of ensemble categories exist: ensembles of different networks and ensembles of variants of the same model.

Ensembles of Different Network Types

In summary, we have four different model types with different performances: P2P, GlassPoseRN, SepConv-LSTM, and SeqCapsPose. This allows us to test the robustness of ensembles to comparably worse members. Since the CapsPose and SeqCapsPose are close in performance and we have to evaluate on the 80/10/10 data split because of SepConv-LSTM, we choose SeqCapsPose for simplicity. Subsequently, only the P2P network had to be retrained on the 80/10/10 data split. The number of models allows us to test how our fusion algorithms behave when they deal with models with significant gaps in accuracy. Therefore, we start by creating ensembles that include the best instance of our four basic network types. Then, we remove the worst model on the 80/10/10 split to fuse only the best three models, and finally, we remove one more model until we are left with only the best two models SepConv-LSTM and SeqCapsPose.

We create three ensembles for each of the combinations, one with an average fusion, one with the weighted average, and the last one using our select best fusion algorithm. All of the non-learning-based ensembles can be generated with relatively little overhead once the fusion algorithms are implemented. In contrast, the learned fusion requires a dedicated and time-consuming training process for each ensemble. Therefore, we used the results from the simple fusions to decide which model combinations we wanted to train the neural network.

Ensembles of Variants of the Same Model for Head and HMD Fusion

To combine HMD and head pose estimation, we utilize the SeqCapsPose model. We combine the SeqCapsPose-HEAD model with SeqCapsPose-HMD to investigate whether this fusion of head and glasses pose can be beneficial. As the SeqCapsPose-HEAD models perform considerably worse, it is not suitable for the average fusion. Hence, we only used the weighted average fusion, which was similar to the learning-based fusion performance when combining different DNNs. In summary, we decided to create and test the following ensembles.

- **Different network types**
  Fusion methods: average, weighted average, and select best fusions for all ensembles, learned fusion for the last ensemble.
  Fused networks:

  - SeqCapsPose, SepConv-LSTM, P2P, GlassPoseRN
  - SeqCapsPose, SepConv-LSTM, P2P
  - SeqCapsPose, SepConv-LSTM

- **Head and glasses pose networks**
  Fusion method: weighted average fusion.
  Fused networks: SeqCapsPose-HMD, SeqCapsPose-HEAD

Having defined which single models we combine into ensembles with which fusion algorithms, they are evaluated in the following.

7.5.2 Neural Network Ensemble Results

In this Subsection, we present and compare the results of the various ensemble methods. After evaluating the fusion results of different AR glasses pose estimation methods, we analyze the fusion of head and glasses pose networks.

Fusion Results of HMD Pose Estimation Methods

The first ensembles we created combined the best instances of all or some of our four selected networks GlassPoseRN, SepConv-LSTM, P2P, and SeqCapsPose. SeqCapsPose has the best estimation performance among them. An ensemble has to achieve higher accuracy than that model in order to improve upon the initial situation. These models have very different accuracies, with GlassPoseRN being the lowest-performing on the 80/10/10 split, followed by P2P, SepConv-LSTM, and finally SeqCapsPose. Since the accuracies of most of these networks are considerably lower than SeqCapsPose, some of the fusions were not promising with regards to improving the pose estimation. However, they offer insight into how well the different fusion algorithms can tolerate models with comparatively inferior estimation performance. In addition, it can be derived how close the model performances are required to be such that a benefit can be expected from including a model with lower estimation accuracy into the ensemble. We started with all models together, then left out GlassPoseRN and then P2P until only the two best models on the 80/10/10 data split, SepConv-LSTM and SeqCapsPose, remained. Table 7.20 shows the resulting error of those ensembles, with the values of the four individual networks in the first four rows for reference. The best values per dimension are highlighted in bold.

None of the fusion models manages to improve the pose estimation performance compared to the SeqCapsPose performance. Nevertheless, the learning-based method, among other non-learning-based examples, comes close to the SeqCapsPose performance.

Average As expected, the average fusion does not perform well when confronted with networks with a significant gap in accuracy. Only when the two best models are averaged, this fusion leads to close errors to the initial SeqCapsPose model.

	Models				MAE Orientation				Translation			
Fusion	GlassPoseRN	P2P	SepConv-LSTM	SeqCapsPose	Roll	Pitch	Yaw	Avg	X	Y	Z	L_2
None	x				2.95	6.01	2.61	3.86	6.47	5.36	6.04	11.95
None		x			1.28	1.04	2.73	1.68	5.14	6.01	2.39	9.47
None			x		0.74	0.72	0.98	0.81	2.67	2.43	1.53	4.46
None				x	**0.23**	**0.35**	**0.34**	**0.31**	**0.97**	**0.87**	**0.50**	**1.60**
Average	x	x	x	x	1.64	1.70	7.54	3.62	2.88	2.88	1.99	5.30
Weighted	x	x	x	x	0.95	1.86	4.66	2.49	1.75	1.63	0.86	2.93
Best	x	x	x	x	0.40	0.44	0.68	0.51	2.02	1.64	1.01	3.18
Average		x	x	x	1.98	1.27	9.30	4.18	2.53	2.69	1.14	4.44
Weighted		x	x	x	0.95	1.79	4.93	2.56	1.66	1.55	0.78	2.77
Best		x	x	x	0.40	0.44	0.68	0.51	2.02	1.64	1.01	3.18
Average			x	x	0.41	0.44	0.67	0.51	1.90	1.70	0.98	3.14
Weighted			x	x	0.42	0.46	0.66	0.51	1.57	1.50	0.77	2.65
Best			x	x	0.40	0.44	0.68	0.51	2.02	1.64	1.01	3.18
Learned			x	x	0.40	0.44	0.68	0.50	1.55	1.53	0.77	2.65

Table 7.20: The MAE for the orientation and the translation results over all dimensions for the best instances of our four different models and the ensembles built from them. The lowest error per dimension are highlighted.

Weighted Average The weighted average performs comparatively well and decreases the error drastically compared to the average, even when all single models are included. However, there is still a very noticeable benefit of leaving out underperforming models. The weighted average ensemble with the best two models performs very similarly to the learning-based fusion of the same models. On some values like the Y-axis and the yaw rotation error, it has an improved performance.

Select Best The main advantage of the select best fusion is that it ignores the predictions from the worst models, making it more robust to outliers. Our results confirm this idea, as all combinations- with the select best have the same results. They reach the same values, no matter which models they combine. This behavior suggests that the select best fusion is best suited for combining only as few as possible models when they have an accuracy gap between them. While the addition of worse models does not impair the result, they also do not benefit the overall result. We conclude that the cancellation of errors by combining multiple predictions arithmetically as done in the weighted average is more useful in our scenario. This is supported by the weighted average results, showing improved results compared to select best on the translation.

Learning-based Fusion Since the learning-based fusion requires a training process, we only used it with the most promising combination of models. The previous results have shown that the two least-accurate models do not add valuable input to the ensemble. Since this fusion type is adaptable due to the training process, it is likely to ignore worse models and focus on better predictions. However, our learned ensemble does perform very similarly to the more straightforward, weighted average fusion.

Inference Time

One fusion step with the learning-based fusion takes 0.851 milliseconds, resulting in more than 1000FPS. Thus, the fusion itself does not add any considerable computation time and does not introduce a bottleneck. The individual networks result in significantly slower inference times due to their comparatively deep architectures. The main computational load in an ensemble comes from having to run multiple individual models. However, we do not see any considerable advantage in fusing the networks.

Results of Fusioning Head and HMD Pose Estimation

Model	MAE Rotation	MAE Translation
Head Pose on Head GT	1.86	7.92
Head Pose with Simple Offset	1.68	4.51
Head Pose with Transformed Offset	1.68	4.12
SeqCapsPose-HMD	**0.31**	1.49
Head-HMD-Ensemble	0.55	**1.43**

Table 7.21: The MAEs of SeqCapsPose models dealing with head and glasses predictions and of the ensembles built from them. The minimum errors for rotation and translation are highlighted.

Before combining the SeqCapsPose-HEAD network estimating the head pose and the SeqCapsPose-HMD regressing the HMD pose, we adapted the head estimation model. The performance of the SeqCapsPose-HEAD on the head pose dataset can be seen in the first row of Table 7.21. We used two different ways to compute an estimate of the HMD position from the head position, as introduced in Section 5.3. The computation of the offset in the orientation stays the same for both approaches. The second and third rows of Table 7.21 reveal that the HMD translation computed with the transformed offset is more accurate than the simple offset alternative. With both means of computing the offset, the resulting HMD pose estimations have lower MAEs in both rotation and translation than the head pose estimations have on the head pose dataset, showing that they work well.

Although our computation of the HMD pose from the head pose works quite well, the head pose model has a generally lower accuracy than the SeqCapsPose-HMD model. We create an ensemble consisting of the head pose model with the transformed offset computation and the standard SeqCapsPose model. Since there is a large gap in accuracy between these models, we deploy the weighted average fusion. Our previous results have shown that it is best suited for this scenario. The last row of Table 7.21 reveals that the ensemble does not improve the orientation prediction. The translation prediction does improve slightly by around 4%. The weights in the weighted average for the SeqCapsPose-HEAD are quite small. This might make them function more as a minor correction in the sometimes right direction instead of a proper pose proposal. It is questionable whether this slight improvement could be reliable enough to consistently work for ensembles with a significant performance gap between models.

7.6 Evaluation in Real-World Conditions

After developing and evaluating numerous 6-DoF pose estimation approaches, we perform a final evaluation in our prototype. We utilize the initial 7 series BMW extended with interior cameras for data collection. We cannot compare with a certain ground truth as there are no markers on the glasses. Hence, we compare the models to each other. In summary, one subject wears the AR glasses used in the dataset, while we record short movement sequences with four selected pose estimation methods. We finally test our algorithms on sunglasses unknown to the networks. Figure 7.7 shows the subject's movement, which was roughly identical in all sequences per glasses model and algorithm.

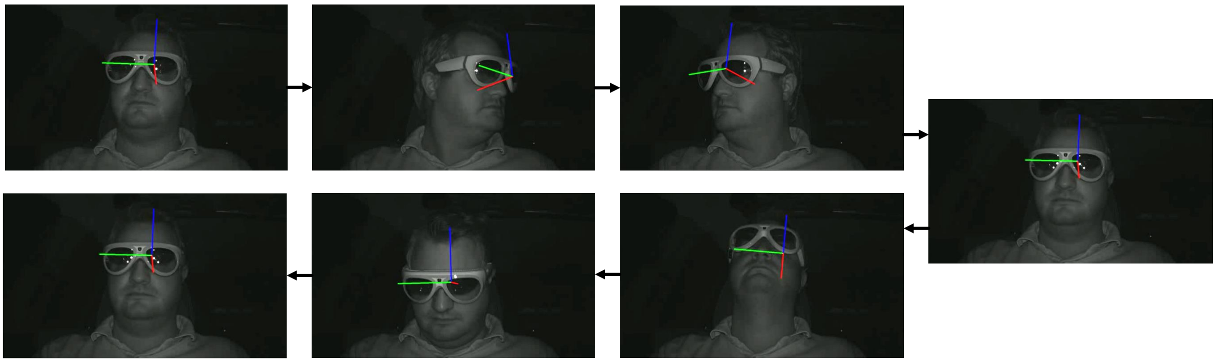


Figure 7.7: Movement of the subject conducted for each glasses model and algorithm. The subject first looks to the front, followed by a head movement from a-pillar to a-pillar. An up and down movement finalizes the sequence.

The sequence starts with the subject looking to the front, followed by a head movement from a-pillar to a-pillar. After returning to the initial pose, an up and down movement finalizes the sequence.

For this evaluation, we select three algorithms. We select the CapsPose, GlassPoseRN, and SepConv-LSTM methods. CapsPose and GlassPoseRN showed the best performances for feed-forward image-based networks. For recurrent tracking methods, we select SepConv-LSTM. We do not deploy SeqCapsPose, as CapsPose performs similarly and is easier to deploy in the car. GlassPoseRN is used with two different variants: one image and three image-based networks. For a better overview, we call the one image network GlassPoseRN1 and the three image network GlassPoseRN3. Each network is tested with individually trained weights per glasses model and the weights ALL, which included all glasses models while training. Thus, we test each network twice per glasses model: with combined weights ALL and with the individually trained weights. We further utilize the ALL weights and the weights trained on the NORTH glasses for test on RayBan sunglasses. This serves to purpose of analyzing the generalizability of the models regarding unknown glasses.

7.6.1 Neural Network Comparison

In our evaluation, we first compare the pose estimation performance of the neural networks trained with the input of a specific glasses model. Afterwards, we analyze their performance when trained on all glasses at once but tested on one specific HMD model.

Evaluation on Specific Glasses Model

To compare the performance of the DNNs trained on the EVS glasses, we plot the pose predictions of all estimators in the same section of the recorded sequences. Figure 7.8 shows the yaw predictions when the subject moves their head from a-pillar to a-pillar and back to the center, wearing the EVS glasses. We omit the plot for position regarding the y-axis, as it results in an almost identical plot of the yaw values. The figure shows that SepConv-LSTM has a rather uneven plot. In comparison, GlassPoseRN1 and especially CapsPose show a smooth prediction. The GlassPoseRN3 plot is smooth until it reaches the second a-pillar. There, the predictions become uneven for that HMD model.

Regarding the upward, z-axis position and the pitch movement of the head, we again observe similar patterns. Since the position estimation along the z-axis shows slightly more amplified errors, we plot the position estimation in Figure 7.9. The plot shows the subject movement where they start in the default position, then move their head up and then down, finally returning to the default position. We can observe that all models have difficulties estimating the position when looking up, as this movement is unusual while driving and was not present during training. The expected movement for this section would be similar to sinus wave, as the subject moves their head up and down, finally returning to the original position. GlassPoseRN3 still performs best when looking up. All other methods result in comparable estimations. When the subject is looking down, SepConv-LSTM again has a comparably rough plot, pointing towards many jumps. All other methods have a mostly smooth shape in that movement section.

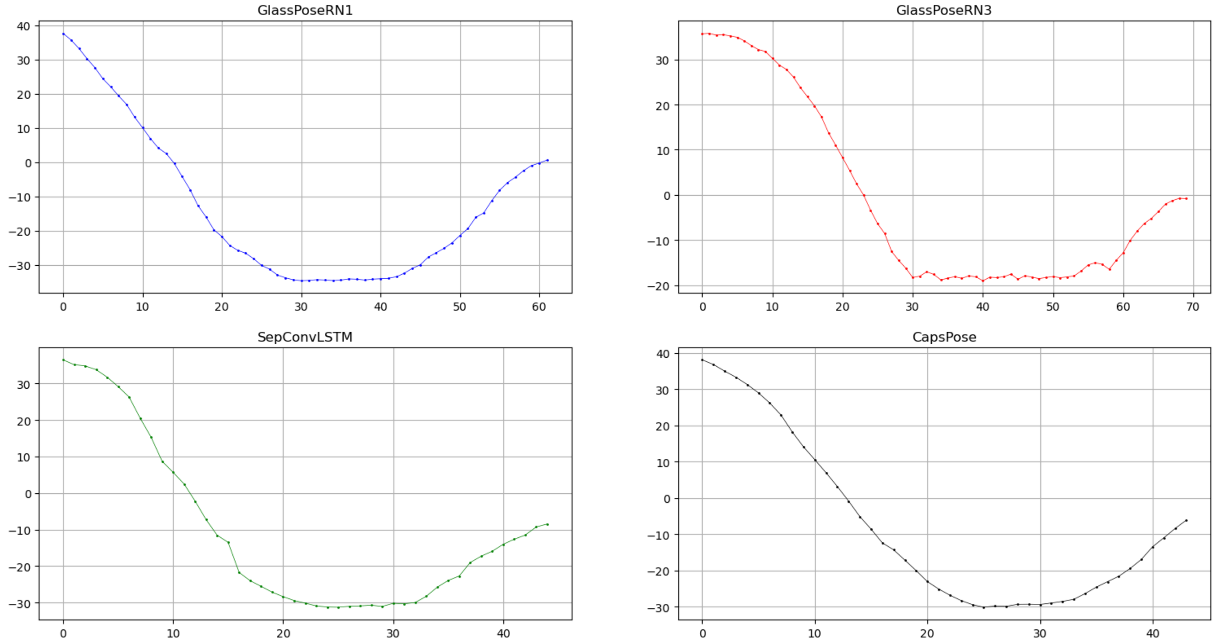


Figure 7.8: A section of the predictions for the EVS glasses of each DNN trained on the EVS glasses only regarding the yaw in degrees.

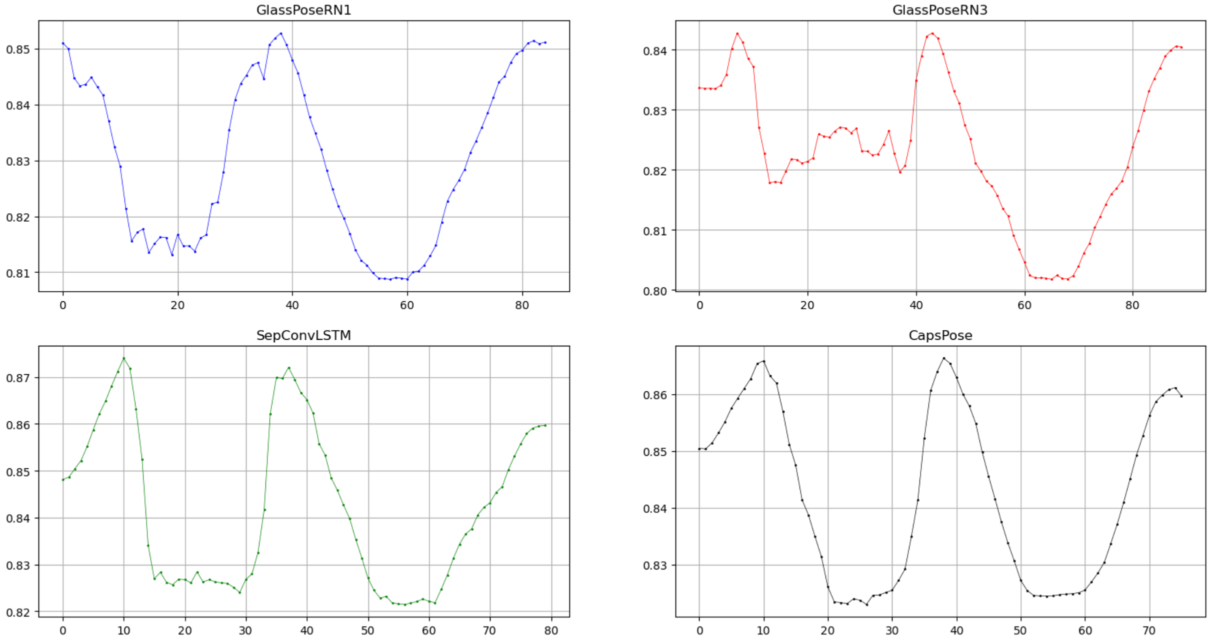


Figure 7.9: A section of the predictions for the EVS glasses of each DNN trained on the EVS glasses only regarding the z position in meters.

Evaluation on Combined Model

We further compare all DNNs on their performance on the example of the ODG glasses, when trained on ALL, combining all recorded glasses types. Figure 7.10 illustrates the yaw estimation per pose estimator, again in the sequence section where the subject moves their head from a-pillar to a-pillar, finally stopping in frontal direction.

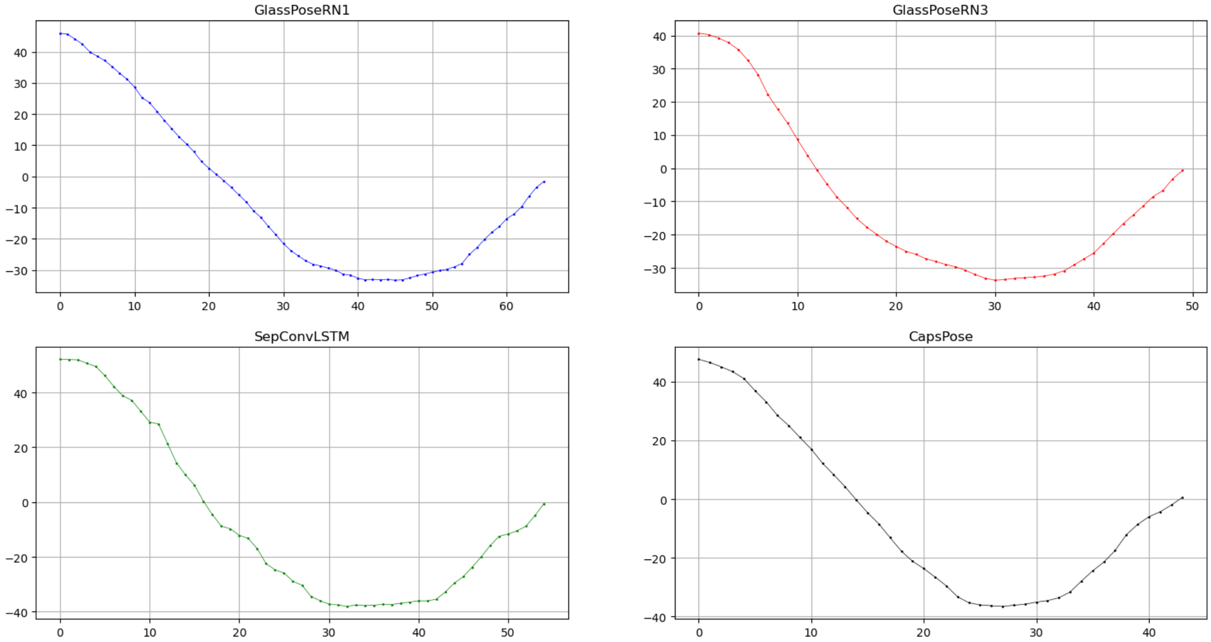


Figure 7.10: A section of the predictions for the ODG glasses of each DNN trained on ALL glasses regarding the yaw in degrees.

SepConv-LSTM again shows a plot with a comparably rough course. Despite GlassPoseRN1 resulting in a mostly smooth graph, GlassPoseRN3 and CapsPose perform best.

Similar to the previous evaluation regarding one specific glasses model, we again plot the z position to evaluate the up and down movement of the subject, displayed in Figure 7.11. The pattern is similar to the performance when trained on a single glasses model. One outlier is SepConv-LSTM, which estimates sensible values when the subject is looking up. CapsPose again estimates quite inaccurate values for the upward movement. The GlassPoseRN variants show comparable performance, with GlassPoseRN3 estimating slightly better values. The downward movement is generally smooth for all methods. Only GlassPoseRN1 is comparably volatile.

In general, even when trained on all AR glasses at once, most models perform well when deployed on one specific HMD. To analyze this observation further, we compare two variants of GlassPoseRN3, trained on HOLO only inputs and ALL inputs, to gather further insight.

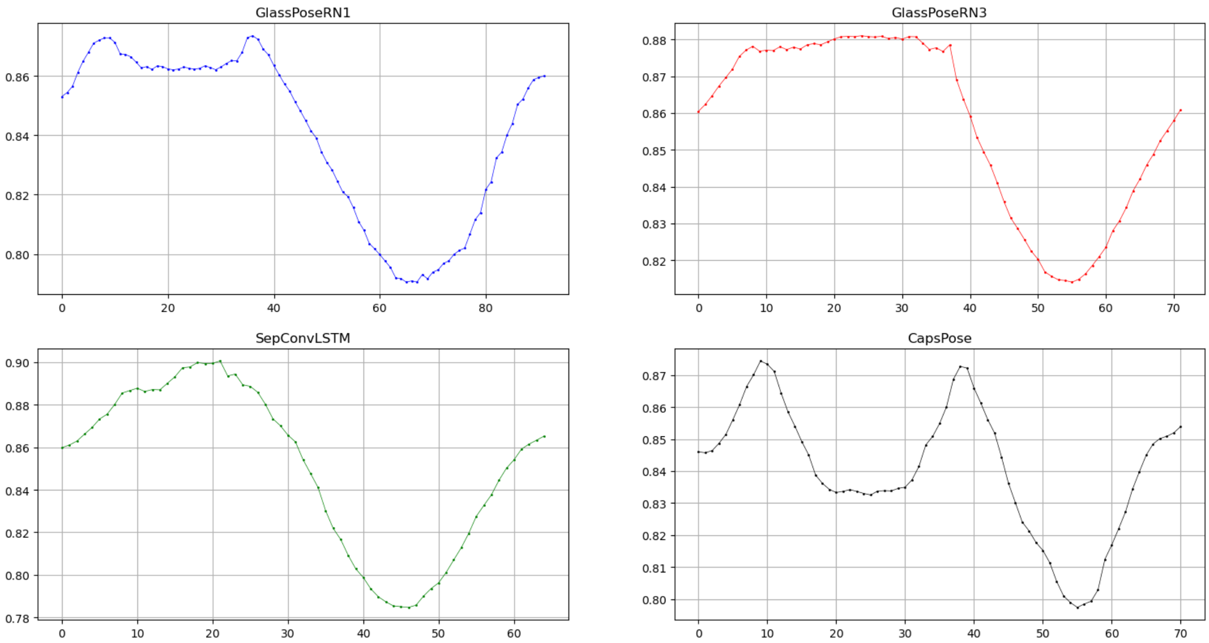


Figure 7.11: A section of the predictions for the ODG glasses of each DNN trained on ALL glasses regarding the z position in meters.

Comparing Combined Models and Single Glasses Models

For comparing the GlassPoseRN3 combined model and single glasses trained model, we choose the HOLO glasses for testing. Figure 7.12 shows the results of the same section as usual for the yaw evaluation.

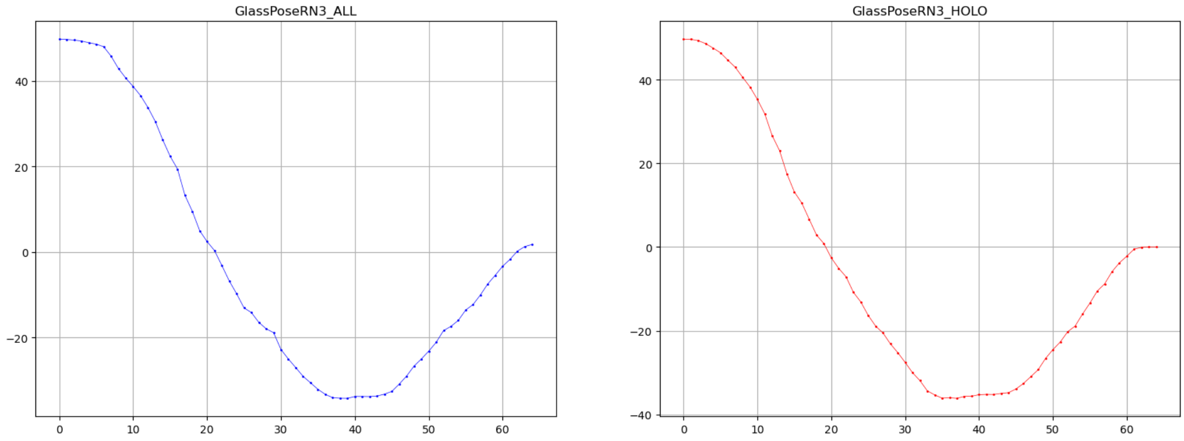


Figure 7.12: A section of the predictions for the HOLO glasses of each GlassPoseRN3 trained with ALL on the left, combining all glasses, and trained on HOLO specifically on the right regarding the yaw in degrees.

We observe very similar patterns in both graphs. The individually trained model shows a slightly smoother course, especially at the beginning of the head turn and during turning. Still, the model trained on ALL shows comparable results.

The graphs regarding the prediction of the upwards axis z again displays similar patterns (Figure 7.13).

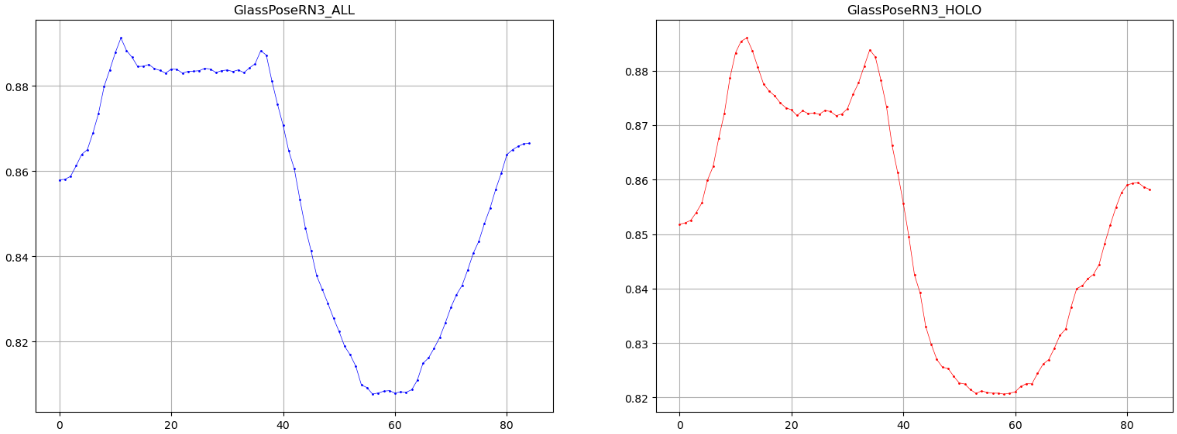


Figure 7.13: A section of the predictions for the HOLO glasses of each GlassPoseRN3 trained with ALL on the left, combining all glasses, and trained on HOLO specifically on the right regarding the z position in meters.

Both models have difficulties predicting the unseen upwards movement. The section where the subject is looking down is registered well by both models. However, the model trained on the specific glasses model only shows a smoother graph.

Generally, we observe slight advantages when using the specifically trained model on a certain HMD. The performance of the model trained on ALL shows that training one model for numerous glasses and later tracking of individual HMDs is possible and shows good estimation performance.

7.6.2 Evaluation on Unknown Glasses

Lastly, we analyze the generalization capability of our DNNs in a real-world evaluation. The aim is to see if the trained models also work on unknown glasses. For this, we again deploy the GlassPoseRN3 model, trained on ALL and the NORTH glasses, respectively. We use the NORTH trained model as they are the most similar pair of glasses to the used sunglasses. For testing, we utilize a RayBan pair of glasses (Figure 7.14).

Figure 7.15 depicts the results of GlassPoseRN3 trained on ALL and NORTH, and tested on the sunglasses. We can see a smooth course on both models. Still, the model trained on ALL results in slightly better estimation performance. It shows marginally superior estimations as it shows a smoother graph.

Similarly, the z position estimation again performs generally well for the sunglasses (Figure 7.16). We see that the model trained on ALL estimates the upward look smoothly. In con-

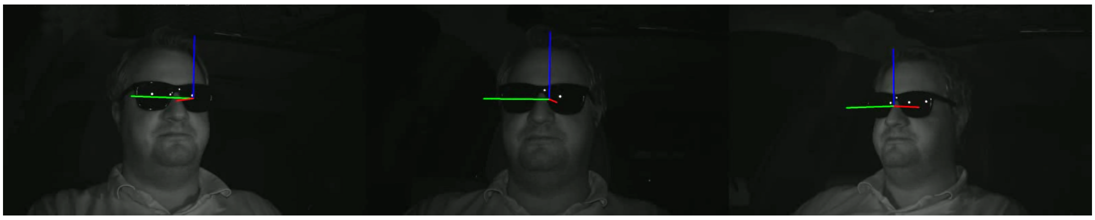


Figure 7.14: The RayBan sunglasses used for the generalization analysis.

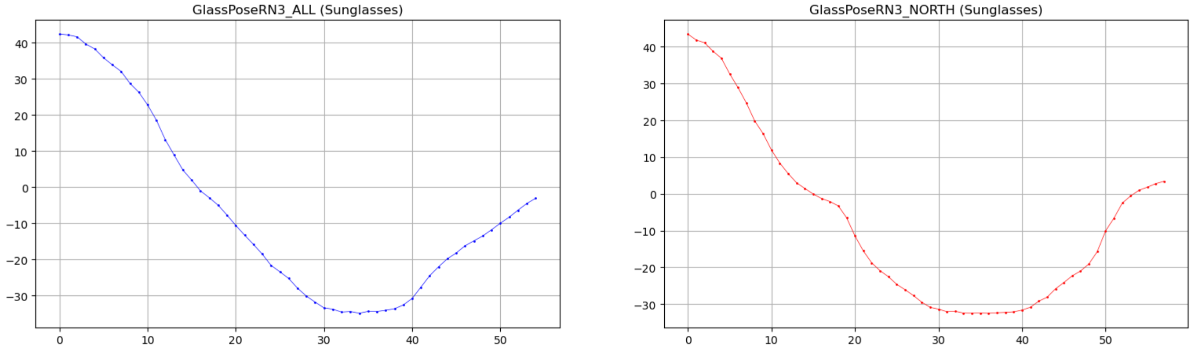


Figure 7.15: A section of the predictions for the sunglasses of each GlassPoseRN3 trained with ALL on the left, combining all glasses, and trained on NORTH specifically on the right regarding the yaw in degrees.

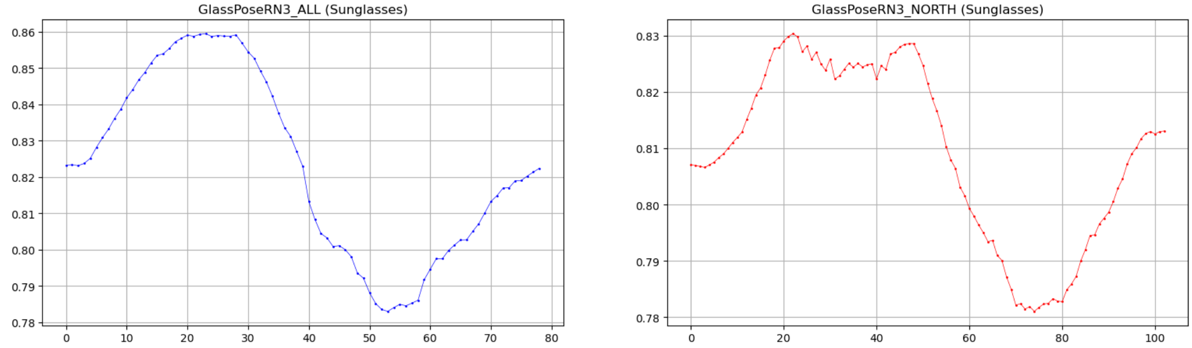


Figure 7.16: A section of the predictions for the sunglasses of each GlassPoseRN3 trained with ALL on the left, combining all glasses, and trained on NORTH specifically on the right regarding the z position in meters.

trast, the model trained on the NORTH glasses has difficulties. Both models perform similarly regarding the section where the subject looks down.

In general, it is observable that the models trained with other types of glasses generalize well to new glasses. The prominent shape of the sunglasses used in the evaluation might help regarding this. The model trained on various glasses models shows better results than the model trained on the most similar sized HMD in the dataset. This points towards an improved generalization performance if a variety of HMDs are used during training, even if the target glasses where not present in the dataset.

7.7 Discussion

After evaluating HMD pose estimation, comparing head and AR glasses pose estimation, evaluating fusion methods, and gaining insight into their real world deployment, we finally discuss them altogether regarding estimation performance and inference time. We also take the initially defined error ranges into account.

7.7.1 Pose Estimation Performance

Our HMD pose estimation methods reach very low errors. Especially the image-based GlassPoseRN and CapsPose methods, and the recurrent image-based methods SepConv-LSTM and SeqCapsPose networks show remarkable pose estimation accuracy. GlassPoseRN reaches 0.1° rotation and 0.9mm position error, while CapsPose lowers it even further to 0.1° orientation and 0.4mm translation error, depending on the data split. SepConv-LSTM estimates the orientation with an error of 0.8° and a translation error of 5.2mm, while SeqCapsPose results in 0.3° for orientation and 1.6mm for position. The point cloud-based method P2P also achieves errors on average around 0.5° in rotation and 5.8mm in translation. In terms of the error ranges defined in Section 3.2, our algorithms fulfill a variety of use case requirements. Table 7.22 shows which method theoretically enables which use case.

	Interior		Car Environment		Distant Object		
Model	Personal Assistant	Button Highlighting	Parking Aid	Mirror Highlighting	Point of Interest	AR Navigation	Lane Highlighting
CapsPose	x	x	x	x	x	x	
SeqCapsPose	x	x	x	x	x	x	
GlassPoseRN	x	x	x		x	x	
SepConv-LSTM	x		x		x		
P2P	x		x		x		

Table 7.22: A selection of our best performing approaches and their fitness regarding the error ranges defined for certain use cases.

CapsPose and SeqCapsPose enable almost all initially defined use cases, including highlighting use cases. They require 0.1 degrees or less if we assume a 5mm position error. The two

methods achieve even lower position estimation error than the assumed 5mm, supporting most highlighting use cases. The only exception is lane highlighting, which constitutes the most demanding use case due to its high accuracy requirement in far ranges. Every other use case, including personal assistant, parking aid, AR navigation, and POI, is possible.
GlassPoseRN achieves comparable orientation error but has a slightly lower position estimation accuracy. Hence, in comparison to the CapsPose models, the mirror highlighting use case seems hardly possible.
SepConv-LSTM and P2P achieve similar estimation accuracies, if the underlying data split used for the methods is neglected. Thus, both support all use cases besides highlighting use cases.

We saw the DNNs performing best on AR glasses pose regression regarding our head and AR glasses pose estimation comparison. The only case when head pose estimation generally performs better is when the head is known to the network during training, but a new HMD is introduced during testing. Head pose estimation also presented potential benefits in extreme poses.

Despite our algorithms already achieving good pose estimation performances, we additionally evaluated potential performance gains through fusing multiple methods. This path seems only worth exploring if a low number of very similar performing methods is considered. In our case, fusing head and AR glasses pose estimation methods exhibited slight improvements regarding the position. However, they require doubled computing resources as the models need to run in parallel. This makes even the slight advantage for head and HMD fusion not practical.

Finally, we evaluated a selection of the algorithms with real, live data from the cameras. In our evaluation, we compared GlassPoseRN with one and three images, SepConv-LSTM, and CapsPose. GlassPoseRN with three images results in marginally better estimation than with one image. Overall, CapsPose and GlassPoseRN with three images perform best. Trainings conducted on all glasses models simultaneously showed comparable estimation performance to models trained on one specific AR glasses model only. This shows the generalization capability of our networks. Even when deploying new pairs of glasses completely unknown to the neural networks, the estimation performs well. Our study on the dataset and the real-world evaluation additionally shows a good performance regarding single image methods.

7.7.2 Memory Requirement & Inference Time

In addition to discussing the pose estimation accuracy, it is important to compare the memory requirement and the inference times of the networks. In a car setting, the memory usage and the inference times are crucial to the selection of the network. Table 7.23 shows an overview of the inference times on GPU and CPU, the memory requirements and the amount of parameters.

CapsPose and SeqCapsPose have close inference times with around 70FPS on CPU and 20FPS on GPU. The inference time on the GPU is higher than on the CPU because the underlying Deep Learning Framework PyTorch. The framework is faster on GPU only in the case of feeding batches into the network instead of single inputs. During real world inference like in-car deployment, a frame-by-frame inference slow the inference down on the GPU. Other networks like GlassPoseRN or SepConv-LSTM are implemented with Tensorflow. SepConv-

Model	FPS on GPU	FPS on CPU	Memory Requirement	Parameters
GlassPoseRN	42	16	10GB	13 million
P2P	5\100	-	2GB	9.9 million
SepConv-LSTM	74	7	10GB	3.8 million
CapsPose	20	70	1GB	5 million
SeqCapsPose	20	70	1.2GB	5.8 million

Table 7.23: The inference times of our networks on the GPU and CPU, their memory requirements and amount of parameters.

LSTM achieves the real-time requirement with a frame rate of more than 74FPS. The GlassPoseRN model also works in real-time with a frame rate of 42FPS. P2P having a preprocessing module that generates the point cloud achieves a very low frame rate on GPU of 5FPS. Without the preprocessing step, P2P achieves 100FPS. P2P cannot be run on the CPU due to its PointNet-backbone.

Regarding the memory usage and number of parameters, SepConv-LSTM consumes more than 10GB of memory for inference with a size of 3.8 million parameters. GlassPoseRN model has a large amount of parameters due to its GlassPoseRN-backbone. It has more than 13 million parameters and consumes more than 10GB. CapsPose uses less than 1GB memory space, but includes around 5 million parameters. SeqCapsPose model has more parameters due to the additional recurrent blocks for the temporal information, using 1.2GB and 5.8million parameters. Despite its high capacity of 9.9million parameters, P2P requires less than 2GB memory space.

Thus, in combination with their outstanding pose estimation accuracies, CapsPose variants and GlassPoseRN are preferable for real world deployment due to their inference times. They both proved their high estimation performances on the dataset, their real-time inference times and their stable performance in the real-world evaluation. CapsPose additionally stands out due to its low memory requirement.

7.8 Summary

This chapter focused on evaluating neural networks for head and AR glasses pose estimation. First, we described the evaluation metrics used throughout the evaluation chapter. Then, the evaluation started with head orientation estimation on the AutoPOSE dataset. This served the purpose of deriving insights for neural network development on IR images.

Based on our findings, we then evaluated the various deep neural networks for AR glasses pose estimation. In a single vs. multi-view pose estimation study, we saw that multi-view images can bring slight improvements regarding image-based pose estimation performance. Moreover, our findings exhibited that our GlassPoseRN and CapsPose methods achieve excellent pose estimation accuracies. Regarding recurrent pose tracking, our SepConv-LSTM and SeqCapsPose algorithms resulted in promising estimation performances. Despite the rudimentary point cloud predictions, point cloud derivation based on infrared images with further pose estimation produced promising results. However, due to the point cloud estimation process, it could not meet real-time inference times.

We then focused on comparing head and HMD pose estimation. For this, we trained several AR glasses pose approaches on the head pose annotations of the HMDPose dataset. In general, HMD pose estimation performance surpasses head pose estimation.

In addition, we evaluated different methods to fuse HMD pose methods or HMD and head pose methods. Our results showed that, in the case fusing head and HMD could slightly increase the translation estimation performance. This was not observable for fusing AR glasses models.

Finally, we deployed a selection of algorithms in the car. This enabled us to evaluate the performance of several methods on images beyond the test set. GlassPoseRN and CapsPose showed remarkable performance on live data.

8 Conclusion

8.1 Summary

This thesis investigates and presents Deep Learning-based in-car 6-DoF AR glasses pose estimation approaches. The goal of the work is an exploration of accurate Deep Learning-based HMD pose estimation.

For this purpose, we first introduce error ranges for Augmented Reality car use cases, acting as target ranges for our pose estimators. We define three categories: car interior, car environment, and distant objects. These categories contain use cases depending on the projection distance relative to the user. The further away a projection is, the more visible an error becomes for the wearer. Moreover, the required error level depends on the use case within the categories: A highlighting use case like button highlighting in the interior requires lower pose errors than an assistant floating on the dashboard.

After introducing error ranges, we present our Deep Learning-based HMD pose estimation algorithms. We first show image-based feed-forward algorithms. Our algorithms range from smaller CNNs like GlassPose to ResNet-based deep CNNs called GlassPoseRN to our more lightweight Capsule-based CapsPose pose estimation network. Secondly, we present numerous pose tracking methods, which leverages the time information present in consecutive frames. The algorithms either include LSTMs only, additional separable convolutions, or Non-Local blocks. A version of CapsPose appended with LSTM layers concludes the tracking networks. Lastly, point cloud-based pose estimation networks conclude the HMD pose algorithms. We derive point clouds based on a semi-supervised point cloud estimation network, only requiring infrared images as input without depth ground truth. The P2P and P2R utilize the point clouds for the AR glasses pose task. As a means to combine the Neural Networks, we introduce several non-learning and learning-based fusion methods.

In the following chapter, head pose estimation approaches are developed and presented. Algorithms trained on the AutoPOSE dataset provide an initial insight into IR-based pose estimation. The learnings gathered from this are used in the HMD pose and further head pose estimation techniques. Thus, most algorithms already presented in Chapter 4 are adapted for head pose in Chapter 5.

For the evaluation of the presented algorithms, we then introduce the recorded HMDPose dataset. It consists of infrared images of drivers wearing different AR glasses models with their 6-DoF pose labels. We acquire head pose annotations from the infrared images for head pose training.

Chapter 7 evaluates all methods on the dataset. First, we assess image-based pose estimation algorithms. Our GlassPoseRN and CapsPose methods achieve real-time performance and outstanding pose results. They attain orientation errors around 0.1° and translation errors around 0.9mm and 0.4mm, respectively. We also show that pose estimation with multi-view image input marginally enhances the pose estimation accuracy. Then, we evaluate image and time-based pose tracking methods. On our dataset, SepConvLSTM and SeqCapsPose run in real-time and show the best results among tracking methods. Further, we analyze pose estimation based on point clouds generated from infrared images. Despite the slow and rudimentary estimation of point clouds, the pose estimators trained on them exhibit good estimation accuracy. Due to the slow depth estimation process, they are not usable in a live setup. When comparing head and HMD pose estimation errors, we observe a generally lower error when using AR glasses annotations and input for training. Fusing various AR glasses pose estimation methods with learning or non-learning-based methods does not improve the estimation accuracy. This finding mainly holds for head and HMD pose combination.

In summary, we present novel and highly accurate AR glasses pose estimators. Estimating the HMD pose only has proven to lead to better results than the head pose alone. A combination of the head and HMD does not further improve the accuracy. Our image-based methods with optional time information incorporation accurately regress the AR glasses pose in real-time. Our test on live data inside the car confirms the excellent estimation results of our algorithms, enabling in-car deployment of AR glasses in the future.

8.2 Future Work

In this thesis, we presented many approaches to estimate the AR glasses pose in the car. However, the research conducted in this thesis led to new questions, which need be addressed in the future.

This work presented various algorithms which run in real-time. Still, the focus was not primarily on improving their inference time but on their pose estimation performance. Thus, it is important to analyze which embedded hardware used in the automotive domain can deploy the algorithms. As embedded hardware typically has limited computation capabilities, tuning or simplifying the networks might be necessary. One option for this would be knowledge distillation in form of teacher student networks. In this case, a smaller network tries to learn the behaviour of a larger network by imitating the outputs of every layer. Another option is to simplify the initial network. To enable a faster inference for the networks, research on potential gains in inference times when deploying tools like TensorRT [114] is required.

AR glasses also enable a variety of passenger use cases, which can be more general entertainment use cases like watching movies or playing video games. Hence, an analysis for multi-person deployment is needed. An evaluation of the case where a network, which receives images from a certain camera angle capturing multiple car passengers, can still estimate the poses accurately in real-time is required. An alternative could be the deployment of a network per camera, while each camera is pointed towards a certain passenger.

List of Figures

1.1 Two widely known Augmented Reality Glasses. 12

2.1 The Reality-Virtuality Continuum. 18
2.2 Head up display of the BMW iX. 20
2.3 AR head up display of the Mercedes S-Class. 21
2.4 Visualizations for in car AR glasses usage. 21
2.5 A standard DNN. 24
2.6 CNN architecture with a classification output. 25
2.7 Comparison of an RNN and a feed-forward neural network. 25
2.8 Overview of the AlexNet architecture. 27
2.9 Overview of VGG16. 27
2.10 Overview of the Inception architecture. 28
2.11 Overview of the ResNet-34 architecture. 29
2.12 Inside-Out vs. Outside-In Tracking. 31

3.1 The two main head-tracking error sources in our AR system. 34
3.2 Overview of the registration error in our setup. 37
3.3 Overview of example use cases and their category. 38

4.1 Overview of the different methods presented in Chapter 4. 41
4.2 The GlassPose algorithm. 42
4.3 An overview of the GlassPoseRN architecture. 44
4.4 The original CapsNet architecture. 45
4.5 Our CapsPose architecture. 46
4.6 Overview of the RvC-based pose estimation method. 48
4.7 The architecture of separable convolution-based CNN-LSTM model. 51
4.8 The architecture of non-local block-based CNN-LSTM model. 52
4.9 The SeqCapsPose network architecture. 54
4.10 An overview of the point cloud estimation architecture. 56
4.11 Mask generation pipeline. 56
4.12 The CNN architecture of the point cloud estimator. 57
4.13 Prediction relative to object box. 59
4.14 Point Cloud prediction examples. 60
4.15 Overview of the P2R-Net architecture. 61
4.16 Overview of the P2P-Net architecture. 63
4.17 Training loss of two networks combining SeqCapsPose and SepConv-LSTM. . 67

4.18 Training loss of the first trainings taking four models as input. 68
4.19 Architecture of the deep fusion network. 69

5.1 The VGG-based CNN used in the POSEidon-framework. 73
5.2 The Head Orientation Network for 128 pixel images. 74
5.3 The ResNetHG18 architecture. 75
5.4 Cropping step used in GlassPose-HMD and GlassPose-HEAD models. 77
5.5 Points class assignment. 80

6.1 The hardware setup on the dashboard inside the car and the car itself. 86
6.2 All coordinate systems and their transformations. 87
6.3 Example, cropped IR images of all four glasses in the recordings. 87
6.4 The simulation setup. 88
6.5 The glasses orientation distribution on all three individual angles in degrees. . . 89
6.6 Example of the Mini Augmented Vision glasses before and after postprocessing. 90
6.7 Visualization of the extracted facial landmarks. 91
6.8 Definition of the head coordinate system. 92
6.9 Distributions of the translation and rotation of a sequence. 94
6.10 Position and orientation distribution per axis of the initial HMDPose labels. . . 96
6.11 Position and orientation distribution per axis of the head pose labels. 97

7.1 Pipeline of the AutoPOSE evaluation. 101
7.2 Comparison by layer amount for ResNetHG and HON. 102
7.3 Overview of three view cases compared on GlassPose and GlassPoseRN. . . . 105
7.4 An overview of our RNN evaluation. 110
7.5 Our pipeline to conduct sequence generation from multi-view IR images. . . . 111
7.6 An overview of the split strategies. 120
7.7 Movement of the subject conducted for each glasses model and algorithm. . . . 130
7.8 EVS predictions of each DNN trained on EVS for the yaw. 132
7.9 EVS predictions of each DNN trained on EVS for the z position. 132
7.10 ODG predictions of each DNN trained on ALL for the yaw. 133
7.11 ODG predictions of each DNN trained on ALL for the z position. 134
7.12 HOLO predictions of GlassPoseRN3 trained on ALL and HOLO of the yaw. . . 134
7.13 HOLO predictions of GlassPoseRN3 trained on ALL and HOLO of the z position. 135
7.14 The RayBan sunglasses. 136
7.15 Sunglasses predictions of GlassPoseRN3 trained on ALL and NORTH of the yaw. 136
7.16 Sunglasses' z position predictions of GlassPoseRN3 trained on ALL and NORTH. 136

List of Tables

1.1 An overview of our novel algorithms and some of their properties. 14

4.1 Size and parameter of CNN-LSTM and SepConv-LSTM. 50
4.2 Parameters for backbone layers. 62
4.3 Overview of input and output for backbone layers. 62
4.4 Overview of all presented and trained algorithms. 70

5.1 The mean offset from head to glasses. 82
5.2 An overview of all presented and trained head pose algorithms. 83

6.1 The different situations on the route and occurring movements. 88
6.2 Inter-Objects translation and rotation deviations between HMD and head. . . . 95
6.3 Intra-Object translation and rotation deviations for the HMD. 95
6.4 Intra-Object translation and rotation deviations for the head. 95
6.5 Inter/Intra-Object translation and rotation deviations between HMD and head. . 96
6.6 Translation and rotation statistics of the HMDPose dataset. 97
6.7 Translation and rotation statistics of the head pose labels. 97

7.1 Results on the 64×64 pixel cropped images of the AutoPOSE dataset. 102
7.2 Results on the 128×128 pixel cropped images of the AutoPOSE dataset. 103
7.3 Direct comparison of all methods on 64×64 and 128×128 resolutions. 103
7.4 Orientation results of GlassPose and GlassPoseRN by view combinations. . . . 106
7.5 Positional error of the GlassPose and GlassPoseRN. 106
7.6 Orientation results of GlassPose, GlassPoseRN, CapsPose and RandomizedBins. 107
7.7 Positional error of GlassPose, GlassPoseRN, CapsPose, and RandomizedBins. . 108
7.8 Rotation results our LSTM-based Pose Trackers. 112
7.9 Results for the positional error of our LSTM-based Pose Trackers. 113
7.10 Frame-by-Frame Pose Estimator and LSTM-based Pose Tracker rotation results. 113
7.11 Frame-by-Frame Pose Estimator and LSTM-based Pose Tracker position results. 114
7.12 Orientation results of the P2R and P2P approaches compared to CloudPose. . . 116
7.13 Results for the positional error of P2P and P2R compared to CloudPose. 118
7.14 Orientation results of feed-forward models for head and glasses pose. 121
7.15 Position results of all models for head and glasses pose estimation. 121
7.16 Orientation results of the best two recurrent models for head and HMD tracking. 123
7.17 Position results of the best two recurrent models for head and HMD tracking. . 123
7.18 Orientation results of CapsPose model for head and glasses pose estimation. . . 124
7.19 Position results of CapsPose for head and glasses pose estimation. 125

7.20 Results of the Ensembles of Different Models. 128
7.21 Results of the Head and Glasses Ensembles. 129
7.22 A selection of our best performing approaches. 137
7.23 The models' inference times, memory requirements and amount of parameters. 139

Bibliography

[1] "Model-Introducing-Google-Glass-Google-Glass-Photos-Collection.jpg (1920×1080)." `https://comingmore.com/wp-content/uploads/2013/05/Model-Introducing-Google-Glass-Google-Glass-Photos-Collection.jpg`, 14.09.2013. *12*

[2] S. Rogers, "What We Know About HoloLens 2," *Forbes*, 11.09.2019. *12*

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. *13, 28, 29, 30, 43, 48, 72, 74, 75, 77, 83*

[4] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic Routing between Capsules," p. 3859–3869, 2017. *13, 30, 45, 47, 48*

[5] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 77–85, 2017. *13, 55, 64, 70*

[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS?17, (Red Hook, NY, USA), p. 5105?5114, Curran Associates Inc., 2017. *13, 32, 61, 80*

[7] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, p. 1735–1780, Nov. 1997. *13, 26*

[8] A. Firintepe, A. Pagani, and D. Stricker, "HMDPose: A large-scale trinocular IR Augmented Reality Glasses Pose Dataset," in *26th ACM Symposium on Virtual Reality Software and Technology (VRST)*, ACM, 2020. *14*

[9] A. Firintepe, A. Pagani, and D. Stricker, "A Comparison of Single and Multi-View IR image-based AR Glasses Pose Estimation Approaches," in *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 571–572, IEEE, 2021. *14*

[10] A. Firintepe, S. Habib, A. Pagani, and D. Stricker, "Pose Tracking vs. Pose Estimation of AR Glasses with Convolutional, Recurrent, and Non-Local Neural Networks: A Comparison," in *EuroXR 2021: Virtual Reality and Augmented Reality*, Springer International Publishing, 2021. *15*

[11] A. Firintepe, C. Vey, S. Asteriadis, A. Pagani, and D. Stricker, “From IR Images to Point Clouds to Pose: Point Cloud-Based AR Glasses Pose Estimation,” *MDPI Journal of Imaging*, vol. 7, no. 5, 2021. *15*

[12] M. Selim, A. Firintepe, A. Pagani, and D. Stricker, “AutoPOSE: Large-Scale Automotive Driver Head Pose and Gaze Dataset with Deep Head Pose Baseline,” in *International Conference on Computer Vision Theory and Applications (VISAPP)*, SCITEPRESS Digital Library, 2020. *15*

[13] A. Firintepe, M. Selim, A. Pagani, and D. Stricker, “The More, the Merrier? A Study on In-Car IR-based Head Pose Estimation,” in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1060–1065, IEEE, 2020. *15*

[14] A. Firintepe, O. Dhaouadi, A. Pagani, and D. Stricker, “Comparing Head and AR Glasses Pose Estimation,” in *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2021. *15*

[15] R. T. Azuma, “A survey of augmented reality,” *Presence: Teleoperators and virtual environments*, vol. 6, no. 4, pp. 355–385, 1997. *17*

[16] N. Tschanz and D. Schart, *Praxishandbuch Augmented Reality: für Marketing, Medien und Public Relations*. Uvk Verlags GmbH, 2015. *18, 19*

[17] P. Milgram and F. Kishino, “A taxonomy of mixed reality visual displays,” *IEICE TRANSACTIONS on Information and Systems*, vol. 77, no. 12, pp. 1321–1329, 1994. *18*

[18] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, “Augmented reality: A class of displays on the reality-virtuality continuum,” in *Photonics for industrial applications*, pp. 282–292, International Society for Optics and Photonics, 1995. *18*

[19] S. Bryson, “Virtual Reality: A Definition History-A Personal Essay,” *arXiv preprint arXiv:1312.4322*, 2013. *18*

[20] P. J. Metzger, “Adding reality to the virtual,” in *Proceedings of IEEE Virtual Reality Annual International Symposium*, pp. 7–13, Sep 1993. *18*

[21] B. Wassom and A. Bishop, *Augmented Reality Law, Privacy, and Ethics*. Elsevier Science., 2014. *18*

[22] M. Tönnis, *Augmented Reality*. Informatik im Fokus, Springer Berlin Heidelberg, 2010. *19*

[23] A. Mehler-Bicher and L. Steiger, *Augmented Reality: Theorie und Praxis*. De Gruyter, 2014. *19*

[24] R. Dörner, W. Broll, P. Grimm, and B. Jung, *Virtual und augmented reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität*. Springer-Verlag, 2014. *19*

[25] B. Furht, *Handbook of augmented reality*. Springer Science & Business Media, 2011. *19*

[26] A. B. Craig, *Understanding augmented reality: Concepts and applications*. Newnes, 2013. *19*

[27] "Der erste BMW iX." `https://www.press.bmwgroup.com/deutschland/photo/compilation/T0333569DE/der-erste-bmw-ix`, 2021-06-29T10:01:15.000Z. *20*

[28] A. Padeanu, "Mercedes S-Klasse (2021) zeigt Mega-Screen und Head-up-Display mit Augmented Reality," 7.7.2020. *21*

[29] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959. *22*

[30] T. Mitchell, *Machine Learning*. McGraw-Hill International Editions, McGraw-Hill, 1997. *22*

[31] R. Dechter, "Learning While Searching in Constraint-Satisfaction-Problems," in *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, p. 178–183, AAAI Press, 1986. *23*

[32] "What are Neural Networks?." `https://www.ibm.com/cloud/learn/neural-networks`, 2021. *24*

[33] V. H. Phung and E. J. Rhee, "A Deep Learning Approach for Classification of Cloud Image Patches on Small Datasets," *Journal of information and communication convergence engineering*, vol. 16, no. 3, pp. 173–178, 2018. *25*

[34] "Recurrent Neural Networks." `https://www.ibm.com/cloud/learn/recurrent-neural-networks`, 14.09.2020. *25*

[35] T. HUANG, "Computer Vision: Evolution and Promise," in *Imaging science and technology, Evolution and promise, 5th International conference on high technology, September 1996*, 1996. *26*

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012. *26*

[37] X. Han, Y. Zhong, L. Cao, and L. Zhang, "Pre-Trained AlexNet Architecture with Pyramid Pooling and Supervision for High Spatial Resolution Remote Sensing Image Scene Classification," *Remote Sensing*, vol. 9, no. 8, 2017. *27*

[38] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *3rd International Conference on Learning Representations, ICLR 2015* (Y. Bengio and Y. LeCun, eds.), May 2015. *27, 31, 42, 73*

[39] J. Jordan, “Common architectures in convolutional neural networks.” `https://www.jeremyjordan.me/convnet-architectures/`, 20.04.2018. *27*

[40] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015. *27, 28*

[41] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157 vol.2, Sep. 1999. *29*

[42] V. Lepetit, P. Fua, *et al.*, “Monocular Model-Based 3D Tracking of Rigid Objects: A Survey,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 1, no. 1, pp. 1–89, 2005. *29*

[43] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, “PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes,” in *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018* (H. Kress-Gazit, S. S. Srinivasa, T. Howard, and N. Atanasov, eds.), 2018. *29, 31*

[44] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, “SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again,” in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. *29*

[45] J. Rambach, C. Deng, A. Pagani, and D. Stricker, “Learning 6DoF Object Poses from Synthetic Single Channel Images,” in *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 164–169, Oct 2018. *29*

[46] B. Tekin, S. N. Sinha, and P. Fua, “Real-Time Seamless Single Shot 6D Object Pose Prediction,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 292–301, June 2018. *29, 30*

[47] M. Rad and V. Lepetit, “BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3848–3856, Oct 2017. *29, 30*

[48] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, “PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. *29, 30, 63*

[49] G. Layher, H. Liebau, R. Niese, A. Al-Hamadi, B. Michaelis, and H. Neumann, “Robust Stereoscopic Head Pose Estimation in Human-Computer Interaction and a Unified

Evaluation Framework," in *Image Analysis and Processing – ICIAP 2011*, (Berlin, Heidelberg), pp. 227–236, Springer Berlin Heidelberg, 2011. *30*

[50] M. Patacchiola and A. Cangelosi, "Head pose estimation in the wild using Convolutional Neural Networks and adaptive gradient methods," *Pattern Recognition*, vol. 71, pp. 132–143, 2017. *30*

[51] N. Ruiz, E. Chong, and J. M. Rehg, "Fine-Grained Head Pose Estimation Without Keypoints," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 2074–2083, 2018. *30*

[52] L.-I. Felea, L. Florea, C. Florea, and C. V. An, "Head Pose Estimation Using Deep Architectures," in *2018 International Conference on Communications (COMM)*, pp. 505–508, IEEE, 2018. *30*

[53] Z. Cao, Z. Chu, D. Liu, and Y. Chen, "A Vector-based Representation to Enhance Head Pose Estimation," in *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1187–1196, 2021. *30*

[54] H. Zhang, M. Wang, Y. Liu, and Y. Yuan, "FDN: Feature Decoupling Network for Head Pose Estimation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 12789–12796, Apr. 2020. *30*

[55] A. Gupta, K. Thakkar, V. Gandhi, and P. Narayanan, "Nose, eyes and ears: Head pose estimation by locating facial keypoints," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1977–1981, IEEE, 2019. *30*

[56] C. Song, J. Song, and Q. Huang, "HybridPose: 6D Object Pose Estimation Under Hybrid Representations," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 428–437, 2020. *30*

[57] B. Chen, A. Parra, J. Cao, N. Li, and T.-J. Chin, "End-to-end learnable geometric vision by backpropagating PnP optimization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8100–8109, 2020. *30*

[58] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects," in *Proceedings of The 2nd Conference on Robot Learning*, vol. 87 of *Proceedings of Machine Learning Research*, pp. 306–316, PMLR, 29–31 Oct 2018. *30*

[59] Z. Li, G. Wang, and X. Ji, "CDPN: Coordinates-Based Disentangled Pose Network for Real-Time RGB-Based 6-DoF Object Pose Estimation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7677–7686, 2019. *30*

[60] S. Zakharov, I. Shugurov, and S. Ilic, “DPOD: 6D Pose Object Detector and Refiner,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1941–1950, 2019. *30*

[61] K. Park, T. Patten, and M. Vincze, “Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7668–7677, 2019. *30*

[62] A. Kendall, M. Grimes, and R. Cipolla, “PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2938–2946, 12 2015. *31, 43, 60*

[63] C. Capellen, M. Schwarz, and S. Behnke, “ConvPoseCNN: Dense Convolutional 6D Object Pose Estimation,” pp. 162–172, 2020. *31*

[64] P. Matthies, B. Frisch, J. Vogel, T. Lasser, M. Friebe, and N. Navab, “Inside-Out Tracking for Flexible Hand-held Nuclear Tomographic Imaging,” in *2015 IEEE Nuclear Science Symposium and Medical Imaging Conference*, (San Diego), 2015. *31*

[65] A. Pinz, M. Brandner, H. Ganster, A. Kusej, P. Lang, and M. Ribo, “Hybrid Tracking for Augmented Reality,” *Ögai Journal*, vol. 21, no. 1, pp. 17–24, 2002. *31*

[66] Delight XR, “Delight XR - XR Glossary.” `https://delight-vr.com/xr-glossary/`, 2018. *31*

[67] R. Wang, Y. Xu, M. A. Sotelo, Y. Ma, T. Sarkodie-Gyan, Z. Li, and W. Li, “A Robust Registration Method for Autonomous Driving Pose Estimation in Urban Dynamic Environment Using LiDAR,” *Electronics*, vol. 8, no. 1, 2019. *32*

[68] L. Liu, H. Li, Y. Dai, and Q. Pan, “Robust and Efficient Relative Pose With a Multi-Camera System for Autonomous Driving in Highly Dynamic Environments,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2432–2444, 2018. *32*

[69] L. Weng, M. Yang, L. Guo, B. Wang, and C. Wang, “Pole-Based Real-Time Localization for Autonomous Driving in Congested Urban Scenarios,” in *2018 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pp. 96–101, 2018. *32*

[70] A. Schwarz, M. Haurilet, M. Martinez, and R. Stiefelhagen, “DriveAHead-a large-scale driver head pose dataset,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–10, 2017. *32, 72, 73, 83, 100*

[71] G. Borghi, M. Venturelli, R. Vezzani, and R. Cucchiara, “POSEidon: Face-From-Depth for Driver Pose Estimation,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. *32, 42, 71, 72, 73, 75, 83, 100*

[72] G. Borghi, R. Gasparini, R. Vezzani, and R. Cucchiara, "Embedded recurrent network for head pose estimation in car," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1503–1508, IEEE, 2017. *32*

[73] T. Hu, S. Jha, and C. Busso, "Robust Driver Head Pose Estimation in Naturalistic Conditions from Point-Cloud Data," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1176–1182, 2020. *32*

[74] G. Farneback, "Very High Accuracy Velocity Estimation using Orientation Tensors, Parametric Motion, and Simultaneous Segmentation of the Motion Field," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 1, pp. 171–177, IEEE, 2001. *32*

[75] R. L. Holloway, *Registration Errors in Augmented Reality Systems*. PhD thesis, USA, 1996. *33, 34, 36*

[76] R. L. Holloway, "Registration error analysis for augmented reality," *Presence: Teleoper. Virtual Environ.*, vol. 6, p. 413?432, aug 1997. *33, 34*

[77] Opencv, "opencv/opencv." `https://github.com/opencv/opencv/tree/master/samples/dnn/face_detector`. *43, 76*

[78] G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, and K. Murphy, "Towards Accurate Multi-person Pose Estimation in the Wild," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3711–3719, 2017. *43*

[79] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. *43*

[80] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020. *43*

[81] A. Berg, M. Oskarsson, and M. O'Connor, "Deep Ordinal Regression with Label Diversity," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 2740–2747, 2021. *48, 70, 107, 108*

[82] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, 2017. *50, 51*

[83] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017. *50*

[84] X. Wang, R. Girshick, A. Gupta, and K. He, “Non-local Neural Networks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7794–7803, 2018. *51, 52, 53*

[85] C. R. Qi, O. Litany, K. He, and L. J. Guibas, “Deep Hough Voting for 3D Object Detection in Point Clouds,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 9277–9286, 2019. *55, 60, 61, 62, 63, 80*

[86] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Weinberger, “Pseudo-LiDAR from Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving,” in *CVPR*, 2019. *55*

[87] C. H. Li and P. K. S. Tam, “An iterative algorithm for minimum cross entropy thresholding,” *Pattern Recogn. Lett.*, vol. 19, pp. 771–776, jun 1998. *56*

[88] C. Li and C. Lee, “Minimum cross entropy thresholding,” *Pattern Recognition*, vol. 26, no. 4, pp. 617–625, 1993. *56*

[89] E. Insafutdinov and A. Dosovitskiy, “Unsupervised Learning of Shape and Pose with Differentiable Point Clouds,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS 18, (Red Hook, NY, USA), pp. 2807–2817, Curran Associates Inc., 2018. *57, 58*

[90] A. Kendall and R. Cipolla, “Geometric loss functions for camera pose regression with deep learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5974–5983, 2017. *60*

[91] A. Schwarz, *Tiefen-basierte Bestimmung der Kopfposition und -orientierung im Fahrzeuginnenraum*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2018. *60, 73, 83, 102, 103*

[92] G. Gao, M. Lauri, Y. Wang, X. Hu, J. Zhang, and S. Frintrop, “6D Object Pose Regression via Supervised Learning on Point Clouds,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3643–3649, 2020. *64, 70, 115, 116, 118*

[93] W. Falcon, “PyTorch Lightning,” *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019. *67*

[94] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-Generation Hyperparameter Optimization Framework,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD ’19, (New York, NY, USA), p. 2623–2631, Association for Computing Machinery, 2019. *69*

[95] M. Selim, A. Firintepe, A. Pagani, and D. Stricker, “AutoPOSE: Large-scale Automotive Driver Head Pose and Gaze Dataset with Deep Head Orientation Baseline.,” in *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and*

Computer Graphics Theory and Applications - Volume 4: VISAPP,, pp. 599–606, 2020. *71, 83, 99*

[96] G. Borghi, M. Fabbri, R. Vezzani, S. Calderara, and R. Cucchiara, "Face-from-Depth for Head Pose Estimation on Depth Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 3, pp. 596–609, 2018. *71, 72, 73, 75, 83, 100, 102, 103*

[97] "ageitgey/face_recognition." `https://github.com/ageitgey/face_recognition`. *71*

[98] A. Newell, K. Yang, and J. Deng, "Stacked Hourglass Networks for Human Pose Estimation," in *Computer Vision – ECCV 2016*, pp. 483–4990, Jan 2016. *72, 74, 83*

[99] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015*, May 2015. *73*

[100] Y. Itoh and G. Klinker, "Interaction-free calibration for optical see-through head-mounted displays based on 3D Eye localization," in *2014 IEEE Symposium on 3D User Interfaces (3DUI)*, pp. 75–82, 2014. *81*

[101] "Explore Focals - North." `https://www.bynorth.com/focals`, 2020. *87*

[102] "Raptor – Everysight." `https://everysight.com/product/raptor/`, 2020. *87*

[103] "MINI Augmented Vision." `https://www.press.bmwgroup.com/global/article/detail/T0212042EN/mini-augmented-vision:-a-revolutionary-display-concept-offering-enhanced-comfort-and-safety-exclusive-prototype-of-augmented-reality-eyewear-underlines-the-innovative-flair-and-creativity-of-the-mini-brand?language=en`, 2015. *87*

[104] "Hololens (1st gen) hardware." `https://docs.microsoft.com/en-us/hololens/hololens1-hardware`, 2020. *87*

[105] A. D. Keedwell and J. Dénes, *Latin squares and their applications*. Elsevier, 2015. *89*

[106] A. Telea, "An Image Inpainting Technique Based on the Fast Marching Method," *Journal of Graphics Tools*, vol. 9, no. 1, pp. 23–34, 2004. *89*

[107] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann, "BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs," 2019. *91*

[108] A. Bulat and G. Tzimiropoulos, "How Far are We from Solving the 2D 3D Face Alignment Problem? (and a Dataset of 230,000 3D Facial Landmarks)," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1021–1030, 2017. *91*

[109] A. Bulat and G. Tzimiropoulos, “Binarized Convolutional Landmark Localizers for Human Pose Estimation and Face Alignment With Limited Resources,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 3706–3714, Oct 2017. *91*

[110] D. W. Marquardt, “An Algorithm for Least-Squares Estimation of Nonlinear Parameters,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963. *91*

[111] M. D. Breitenstein, D. Kuettel, T. Weise, L. van Gool, and H. Pfister, “Real-time face pose estimation from single range images,” in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2008. *92*

[112] B. Iglewicz and D. C. Hoaglin, *How to detect and handle outliers*, vol. 16. Asq Press, 1993. *93*

[113] F. A. A. Garcia, “Tests to identify outliers in data series,” *Pontifical Catholic University of Rio de Janeiro, Industrial Engineering Department, Rio de Janeiro, Brazil*, vol. 50, 2012. *93*

[114] GitHub, “GitHub - NVIDIA/TensorRT: TensorRT is a C++ library for high performance inference on NVIDIA GPUs and deep learning accelerators.” `https://github.com/NVIDIA/TensorRT`, 05.10.2021. *142*

Curriculum Vitae

Ahmet Firintepe

Education

2011 - 2014 — **Ludwig-Maximilians-Universität München**
Munich, Germany
Bachelor of Science, Media Informatics

2017 — **The University of Sydney**
Sydney, Australia
Study Abroad

2014 - 2017 — **Ludwig-Maximilians-Universität München**
Munich, Germany
Master of Science, Computer Science

Professional experience

2016 — **BMW AG**
Munich, Germany
Intern

2016 - 2017 — **BMW AG**
Munich, Germany
Master Thesis Intern

2018 - 2021 — **BMW AG**
Munich, Germany
Researcher

Publication list

Author's publication list as of November 2021

Journal articles

1. Ahmet Firintepe, Carolin Vey, Stylianos Asteriadis, Alain Pagani, and Didier Stricker. From IR Images to Point Clouds to Pose: Point Cloud-Based AR Glasses Pose Estimation. *MDPI Journal of Imaging*, vol. 7, no. 5, 2021.

Conference papers

2. Mohamed Selim, Ahmet Firintepe, Alain Pagani, and Didier Stricker. AutoPOSE: Large-Scale Automotive Driver Head Pose and Gaze Dataset with Deep Head Pose Baseline. In *International Conference on Computer Vision Theory and Applications (VISAPP)*, SCITEPRESS Digital Library, 2020.

3. Ahmet Firintepe, Mohamed Selim, Alain Pagani, and Didier Stricker. The More, the Merrier? A Study on In-Car IR-based Head Pose Estimation. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1060–1065, IEEE, 2020.

4. Ahmet Firintepe, Alain Pagani, and Didier Stricker. HMDPose: A large-scale trinocular IR Augmented Reality Glasses Pose Dataset. In *26th ACM Symposium on Virtual Reality Software and Technology (VRST)*, ACM, 2020.

5. Ahmet Firintepe, Alain Pagani, and Didier Stricker. A Comparison of Single and Multi-View IR image-based AR Glasses Pose Estimation Approaches. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 571–572, IEEE, 2021.

6. Ahmet Firintepe, Sarfaraz Habib, Alain Pagani, and Didier Stricker. Pose Tracking vs. Pose Estimation of AR Glasses with Convolutional, Recurrent, and Non-Local Neural Networks: A Comparison. In *EuroXR 2021: Virtual Reality and Augmented Reality*, Springer International Publishing, 2021.

7. Ahmet Firintepe, Oussema Dhaouadi, Alain Pagani, and Didier Stricker. Comparing Head and AR Glasses Pose Estimation. In *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2021.