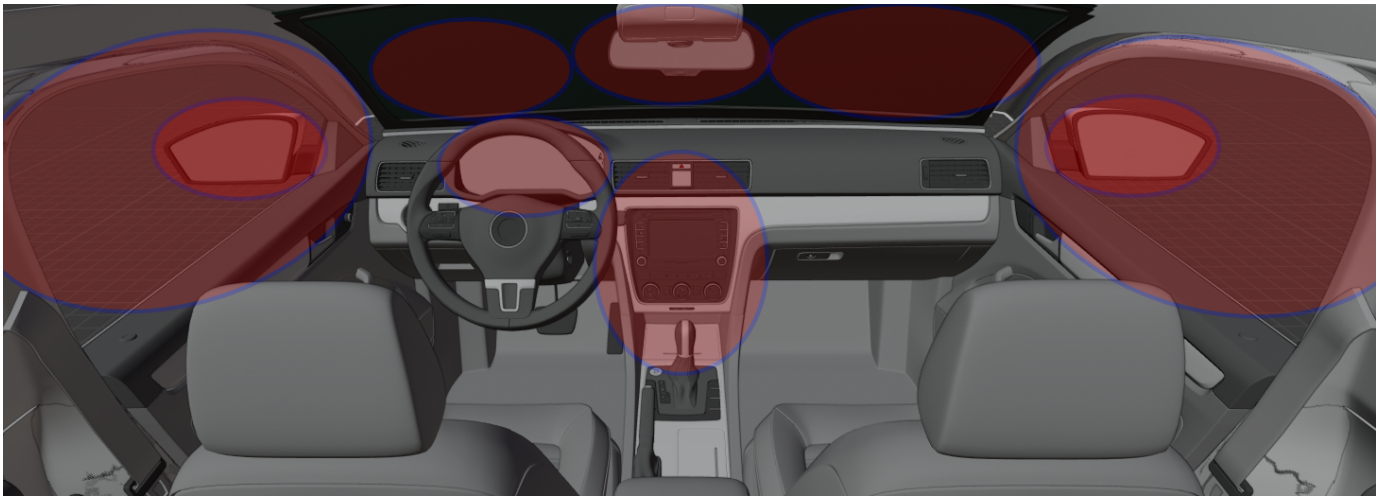
CHALMERS
UNIVERSITY OF TECHNOLOGY



Adaptive Gaze Sector Analysis

Clustering of Automotive Interior Gaze Data without Explicit Calibration

Master's thesis in Systems, Control and Mechatronics

VICTOR BRANDT

CHRISTOFFER HANSSON

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Adaptive Gaze Sector Analysis

Clustering of Automotive Interior Gaze Data without Explicit Calibration

VICTOR BRANDT, CHRISTOFFER HANSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2020

Adaptive Gaze Sector Analysis
Clustering of Automotive Interior Gaze Data without Explicit Calibration
VICTOR BRANDT
CHRISTOFFER HANSSON

Supervisors: Henrik Lind, Torsten Wilhelm, Smart Eye AB
Examiner: Karinne Ramirez-Amaro, Department of Electrical Engineering

Master's Thesis 2020
Department of Electrical Engineering
Division of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Rendering of the interior of a car with ellipses overlaid corresponding to the center and covariance of each cluster.

Typeset in LaTeX
If you are holding this thesis in your hand, it was printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Adaptive Gaze Sector Analysis
Clustering of Automotive Interior Gaze Data without Explicit Calibration
VICTOR BRANDT
CHRISTOFFER HANSSON
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Driver monitoring systems provide increased safety in vehicles by studying the face of the driver and drawing conclusions about the driver's attention and focus. An important feature of driver monitoring systems is the ability to accurately calculate which areas or objects inside a car drivers look at. Objects that a driver could be looking at are for example the rear mirror, the center entertainment stack, or the left window. Many driver monitoring systems rely on a calibration step ahead of a driving session to provide an accurate gaze measurement. However, such a step needs to be repeated any time conditions change, such as for every new driver, which is not always feasible or desirable in a real-life implementation in a production vehicle. Further, in many driver monitoring systems, the driver's gaze is not always visible to the camera, as many production systems only contain a single camera setup. This thesis presents a solution to these problems that incorporates a mix of a regression model and unsupervised learning. Input data in the form of gaze direction and head pose direction from a single-camera system is used to approximate the current gaze direction. This new gaze direction is clustered using a modified version of the expectation-maximization clustering algorithm. The clustering algorithm is then used to classify the gaze point. The result is an algorithm that adapts to an unknown user and calculates the best possible gaze direction and finally uses this information to find the most likely object that the driver is focusing on. The results show that the final algorithm can correctly classify $\approx 35\%$ of the gaze measurements using a single-camera system, which is only $\approx 0.66\%$ less than using a multi-camera system. Although the accuracy has the potential to improve, this still shows that the proposed solution can handle lower quality data from a single-camera system and still provide similar results as when working with higher quality data from a multi-camera system. As no previous work for a self-calibrated algorithm applied to provide a gaze sector indication in a vehicle setting has been found by the authors, this thesis is deemed a good starting position for further development.

Keywords: Gaze Sector Analysis, Clustering, Unsupervised Learning, Driver Monitoring Systems, Multi-layer Perceptron Regression

Acknowledgements

We would like to thank everyone at Smart Eye AB in Gothenburg for welcoming us and showing us how a good cup of Italian coffee is made. A special thanks to our supervisor Henrik Lind and everyone at the Research department for making us feel like a part of your team and for helping us grow as data scientists.

Our greatest thanks go out to Torsten Wilhelm for patiently guiding us through this process, dissecting any and all of our infeasible ideas, and helping us wipe the dirt off any gold nuggets we stumbled upon. This thesis would not have been possible without your help.

We also want to thank our examiner at Chalmers, Karinne Ramirez-Amaro, for giving us excellent feedback. We could not have wished for better academic support than what you gave us.

Victor Brandt & Christoffer Hansson, Gothenburg, May 2020

Contents

List of Figures

List of Tables

1
Introduction

Vehicle safety has been undergoing a massive change in the past decade. This field has historically been dominated by *passive safety*, an umbrella term for techniques that aim to minimize the harm done to passengers during an accident, such as safety belts and airbags. Presently, vehicle safety is expanding into the area of *active safety*. In contrast to passive safety, active safety looks to prevent car accidents from happening in the first place. This area of technology includes not only control of traditional components such as brakes and headlights but also advanced driver assistance systems that rely on software. The latter is a cornerstone to achieve safe autonomous vehicles.

The organization Society of Automotive Engineers has defined a six-level scale for autonomous driving [1], spanning from no autonomy (level 0), to fully autonomous driving (level 6), which does not require any human input. Although such fully autonomous vehicles are still facing legal, ethical, and technological obstacles, we are seeing lower level, partially autonomous driving systems being incorporated into such active safety systems that work in cooperation with a human driver to prevent accidents. An example of this could be a situation where the active safety system receives sensor input that the vehicle in front is braking sharply while the driver of the subject car is inattentive to the road ahead, and in response the active safety system engages the brakes to avoid a collision.

In the example above, the autonomous system concluded that the driver was distracted from the road and acted based on that. Had the driver been paying attention to the road, the autonomous system could have acted differently, for example by triggering a warning to the driver and allowing the driver to make a decision themselves. This example highlights the importance of driver monitoring systems making accurate conclusions about the attention of the driver, as falsely concluding that the driver is attentive to the road ahead could result in an accident.

1.1 Background

Driver monitoring systems (DMS) are increasingly becoming more common as consumer demand and legislation are pushing for this functionality to be implemented. The Parliament of the European Union [2] and the Council of the European Union [3] has approved legislation that mandates all vehicles to be fitted with advanced safety systems, including driver monitoring systems, by the year 2022. As over 15 million

new car registrations occurred in the European Union in 2019 [4], this new legislation exemplifies the growing need for a driver monitoring system that is reliable and convenient to use for consumers.

The company Smart Eye AB has developed a state-of-the-art driver monitoring system for integration in passenger cars and other vehicles, currently in production in six vehicle models [5]. This DMS facilitates better safety as well as functions that improve the user experience. The DMS contains eye-tracking, artificial intelligence software which, by studying a person's eye, face, and head movements, can draw conclusions about a person's alertness, attention, and focus. The DMS is available both for multi-camera systems (multiple cameras observing the driver's face from different angles) and single-camera systems.

In any high-performance gaze measurement systems, for example the Smart Eye DMS, an initial gaze calibration step must be performed to identify the exact eye geometry of the person using the system. This explicit calibration step typically requires the driver to explicitly look at a series of specific coordinates in the car, spending a few seconds looking at each point. Through this, a precise 3D gaze direction vector can be calculated based on the positions of the pupil and glints (corneal reflections from infrared light sources with known positions) and this gaze vector can be intersected with objects of a 3D model of the world to estimate the driver's direction of attention. Knowledge of the driver's attention can then be used to prevent accidents, such as the one in the example above.

However, driver monitoring systems such as the Smart Eye DMS still face several problems:

- An explicit gaze calibration step is neither always feasible or desirable. The system has to adapt to an unknown user and calculate the best possible gaze direction.
- Precise gaze includes noise factors from refraction in glasses and contact lenses.
- If the DMS uses only one camera, then exact 3D measurements are not possible to obtain (follows from the fact that a single 2D image is insufficient to represent a 3D measurement).
- The driver's eyes are not fully visible to the camera, e.g. when the driver is looking towards the outer mirrors.

With these problems in mind, it appears that there exists a demand in the automotive industry for a solution for driver monitoring systems that is able to autocalibrate (i.e., does not need an explicit calibration step), works for a single-camera system and can estimate the gaze direction even when the eyes are not fully visible.

1.2 Objective

The purpose of this master's thesis is to enhance the functionality of the company Smart Eye's eye-tracking algorithm in the area of gaze sector analysis. More specifically, the objective is to determine what area of the vehicle interior the driver is

looking at. This is an essential step towards a higher level of automation and interactive environment in the interior of vehicles, as well as creating a more robust driver awareness observation to aid the active safety functions that are growing in both popularity and capability in today's market. Such active safety functions include tracking dangerous behavior while driving, for example handling a smartphone or falling asleep, and acting to prevent such dangerous behavior from causing accidents [5].

Due to requirements from Smart Eye's original equipment manufacturer clients, it is not possible to expect a calibration session to be performed ahead of each driving session. Therefore, in order to estimate the visual attention of the driver, the software must be automatically calibrated while driving. As a result, this thesis focuses on finding methods of creating a general attention model without requiring the driver to explicitly calibrate the software to achieve sufficient accuracy to detect the driver's attention. Instead, calibration will be performed with data from the camera(s) without specifically prompting the driver to take any action to calibrate; we call this *implicit calibration*.

Further, the resulting algorithm needs to work with a single-camera setup as this is the setup used in the production vehicles of Smart Eye's customers. Contrary to the multi-camera setups found in Smart Eye's testing-vehicles, these single-camera systems provide a substantially lower measurement accuracy than their multi-camera counterpart, especially when the head of the driver is facing away from the camera. Therefore, the resulting algorithm needs to work with Smart Eye's single-camera system, and use the available data to extract a comparable sector prediction to the one obtained using the multi-camera setup.

Lastly, as the objective is to determine what general area in the car the driver is looking at, it is not necessary to know the exact position of the intersection between the driver's gaze direction and the objects in the car. Instead, the determining of what object is currently being looked at will be done by classifying the gaze data to provide a probability vector that contains the probability that, at a certain time frame, the drivers is looking at a certain object. Here, an object refers to for example the side mirror, the center entertainment stack, or the speedometer.

1.3 Implementation

The problems described will be solved in steps. First, a method of estimating the gaze direction is developed. This method will be called whenever an accurate calculation of the actual gaze direction is not possible, for example due to the driver's eyes not being fully visible to the camera. Next, the gaze direction (and estimated gaze direction) will be classified to determine what object in the car the driver is looking at. The gaze direction data will be grouped using a clustering method. This clustering model will be initialized to fit the geometry of the interior of the car, and then be continuously updated as the driver operates the vehicle and provides new data.

The proposed methods for solving these problems will be introduced in chapter 2. Following this, the results from these proposed methods will be presented and discussed in chapter 3, after which a final algorithm proposal containing some of the proposed methods will be presented and tested in chapter 4. This structure can be seen in Figure 1.1. Lastly, a conclusion regarding the results of the thesis, as well as the performance and takeaways of the proposed algorithm will be included as chapter 5.

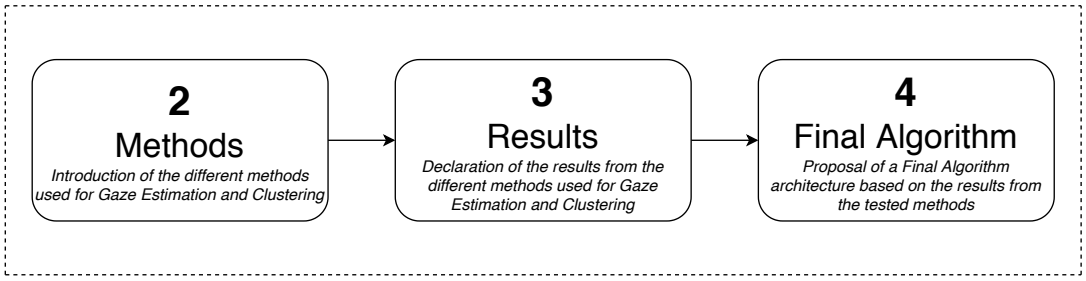


Figure 1.1: Flowchart of the main chapters of this thesis

During the process of this thesis, Victor Brandt was mainly responsible for chapters 2.2, 2.3.3, 2.3.4, 3.1, 3.2.1, 3.2.2, 3.3, 4.

Christoffer Hansson worked on parts of the project explained in the following chapters: 2.1.1, 2.1.3, 2.1.4, 2.1.5, 2.3.1, 2.3.2, 2.3.5, 2.3.6, 2.4.1, 2.4.2, 2.4.3, 3.4, 4.1.

1.4 Limitations

Much of the groundwork when it comes to the software of the Smart Eye DMS had already been done and was at the time of writing the thesis in use by Smart Eye. This leads to the scope of this project incorporating the essential functionality described in this section, and not the elementary functionality of the eye-tracking system. Moreover, a substantial data collection had already been done by Smart Eye, and no further data needed to be collected. This data along with a 3D CAD model of the car had already been implemented in existing scripts that were available to project and visualize the existing data. These scripts were used and modified for projection and visualization, to allocate as much time as possible on the functional parts of the algorithm development.

The main focus of this thesis is highway driving as opposed to driving in an urban setting. In terms of complexity, the difference is that driving on a highway generally requires less head motion and gives more predictability in terms of what sector the driver focuses his or her attention on. Once highway driving has been sufficiently solved, meaning that the results are of the algorithm are satisfactory, the next natural step is to continue to incorporate driving in an urban environment. This would, however, be outside the scope of this project. Further, the software that this project results in needs to be able to produce satisfactory results with a

one-camera system, as this more closely resembles the product that most customers of Smart Eye use. However, the more accurate data from the multi-camera system was used to build an offline model which is incorporated in the algorithm that the single-camera system uses. This multi-camera setup data was also used as a performance baseline for testing, as the goal is to obtain similar performance to the multi-camera system using only a single-camera setup.

The project will be limited to developing a method for separating and classifying of the gaze sector, and will thus not examine or develop anything related to the functionality or usage of the results.

To summarize, this thesis is limited to

- Developing a gaze sector classifier, rather than calculating a precise gaze direction
- Only using data recorded during highway driving
- Operating on single-camera live data, however, training may include multi-camera data

This master's thesis was developed under a non-disclosure agreement.

1.5 Ethical and Sustainability Considerations

This project aimed to enhance Smart Eye's eye-tracking algorithm which is used in cars to provide safety features while driving, as well as enhancing the user experience of the car. The increased safety that follows from this contributes to the United Nations' (UN) Sustainable Development Goal #3 [6], where one of the sub-goals is to *reduce the number of deaths and injuries from road traffic accidents*. The outcome of this project can thus be seen as supporting this Sustainable development goal.

On the other hand, the features that enhance the user experience but provide no added safety could be seen as extravagant, and thus perhaps could conflict with the UN goal #12 [6] that relates to *responsible consumption and production*. These features are not something that adds value in accordance with any of the UN Sustainable Development goals, while at the same time theses features might increase the desirability of the product and thus might create an increased demand for a new car that contains these features. This would lead to an increase in material and energy consumed, which is in conflict with UN goal #12.

As the results from this thesis are not yet ready for commercial use, the actual impact is yet to be determined.

1.6 Related Work

The first found work considered to incorporate the field of cluster analysis was published nearly a century ago [7] and offered an approach to classify cultural aspects in

the field of anthropology. Since then different methods of clustering have been developed for classification and optimization purposes. The applications can be found in almost any field of study where distinguishing features are present, and more recently it has become one of the building blocks which data mining heavily relies on.

The interest in machine learning has grown in recent times. Machine learning algorithms generally fall into one of two categories; supervised and unsupervised. Supervised algorithms are trained using labeled data, i.e. the desired result is known ahead of training, and a ground truth to compare against can easily be obtained. Supervised algorithms include backpropagation and artificial neural networks. Clustering is instead an example of an unsupervised type of algorithm. Unsupervised learning does not work with labeled data, and is thus unaware of the desired outcome. Instead, it looks for similarities in the input data and attempts to group the data accordingly. In other words, supervised learning is aware of the expected result and tries to find a model that returns output close to the expected while unsupervised learning is unaware of what the true result should be and instead tries to find a model based on the correlation of the data. Clustering is therefore a way of categorizing unlabeled data based on the differences and similarities between all the samples, to be able to draw conclusions from the information within the data set.

Using clustering methods is valuable when the size of a given data set is large, and the time required to manually annotate the data is not worth the slight increase in accuracy because of the type of problem that is faced. Assessing differences and similarities between data is an example of a well-suited problem for unsupervised learning in general, and clustering especially. It should be emphasized that more accurate results could have been obtained, had the data been properly labeled. However, in many real-life applications, labeled data is unavailable.

The literature available regarding methods of estimating low-quality gaze data and the clustering of this without prior calibration, that can be directly applied to this project, is limited. Some research has been done on how to utilize a one-camera system [8] in a vehicle setting to estimate and cluster gaze angles, however, this is primarily done through investigating how to extract feature vectors and gaze vectors, while largely ignoring how these interplay with the model of the car, for example by identifying what sectors the driver's gaze is focusing on. There has also been work on gaze patterns in train drivers [9]. In this work, however, the drivers wore an eye tracking device, meaning that the issue of losing gaze data due to facing away from the camera was avoided. A useful resource was the work done by Lee et al. [10], which studied how gaze in a car environment can be estimated in real-time. This work, however, assumes that the gaze direction has been explicitly calibrated ahead of running the algorithm.

Also, the literature on applications within cluster-analysis of images has been quite one-sided, where most of the applications of clustering have been used for helping with categorizing segmented objects from semantic segmentation, such as [11]

and [12]. Applying a clustering algorithm to eye-tracking data to predict what object a person is looking at is therefore quite a new field with high potential, which is why this thesis aims to provide a good base for further work.

Calibration of gaze direction has been shown to have a significant impact on accuracy. In [13], Krafka et al. studied the impact of calibration for gaze tracking while using a mobile phone and tablet, and found that the average error distance was lowered by 22% and 17% respectively after calibrating on 13 points compared to no calibration.

2
Methods

The purpose of the following sections is to introduce the main building blocks that make up this thesis, namely the gaze estimation (section 2.2) and the clustering algorithm (section 2.3). The gaze estimation section describes the different approaches that aimed to provide information to where the driver of the vehicle is looking based on the head pose direction while the gaze tracking quality is sub-optimal. The clustering algorithm section describes the methods used to relate this resulting gaze data (both measurements and estimations) from the estimator to sections of the vehicle interior. Section 2.1 aims to explain how the data was structured, analyzed, and prepared for usage in these latter sections, as well as which simplifications and assumptions were made based on these analyses.

2.1 Data Analysis

A substantial amount of data in the form of driver monitoring video recordings were available to use in this project. This data consists of 15 different individuals providing circa 30 minutes of driving data each (see table 2.1). Two algorithms, one for multi-camera systems and one for single-camera systems, developed by Smart Eye has been run on these videos to create two different log files for each video. The cameras work at around 60 fps and each log file consists of about thirty minutes of driving, resulting in each file containing about 100,000 data points. Each line of data consists of around 175 columns, of which the columns tracking gaze direction and head pose in Euler angles were of most importance to this project.

The multi-camera system, also known as *Smart Eye Pro* or *SEP*, consists of five cameras aimed at the driver from different angles. The single-camera system is known as *Tracker Core* or *TC*. The placement of the cameras are shown in Figure 2.1. As the SEP system is able to view the driver's face from more angles, it also achieves higher accuracy than the TC system in tracking the driver's gaze. As a reminder, this project looked to developed an algorithm that is successful in estimating what object the driver is looking at using the TC system. However, it is only required that the system works for the TC system while running online. As the SEP data has been made available for this project, its higher accuracy has been used to train a model offline and to find correlations between the output from the TC system and the SEP system. It has also been used as a performance baseline to compare how well the proposed algorithm can make the TC data provide similar performance as the SEP data for the sector analysis.

ID	Length of data (no. measurements)
id5	99613
id9	103012
id12	95336
id19	79569
id29	91107
id32	108593
id51	102846
id56	99842
id57	101570
id59	90622
id60	121462
id61	101570
id64	90622
id76	23564
id86	87088

Table 2.1: All data files that were available for training and evaluation in this thesis. Each id-number represents a specific individual.

In order to evaluate the performance of the algorithms, a ground truth had to be defined, to which the tested algorithms' performance were compared. The most accurate method available during this project was to find the 3D gaze direction from the multi-camera system and calculate what object in the car this intersects with. The objects were defined in the CAD model of the car used for testing, and from this the object that the driver is looking at could be determined by finding the intersection point from the gaze vector. However, due to factors such as low gaze quality (see Chapter 2.1.3) and the gaze cone (see Chapter 2.1.4), this ground truth often contains inaccuracies in regards to what object the driver might be looking at. Due to the lack of a fully accurate annotation on which object each data point is correlated to, it is not possible to exactly assess the accuracy of this ground truth. However, from manual inspection of the ground truth and its 3D intersections being at least a gaze cone diameter from the border of each object, it was found that the accuracy was $\approx 60 - 75\%$.

This sub-chapter details how the available data was used to extract useful information about the drivers' gaze direction and head pose.

2.1.1 Areas of Interest

Detecting every single object in the vehicle interior that a driver might interact with or look at during driving is a daunting task, and the demands on the system to be able to do this with accuracy is not possible using the technology available today. Such accuracy, however, is not necessary when it comes to functionality. It is not exceedingly relevant to be aware of all objects a driver might look at, but moreover, the most important general areas of the interior are useful to obtain information about. This information can be useful for different functionality applications in ac-

tive safety and comfort of use.

With that in mind, this thesis has chosen to divide the vehicle interior into nine different sectors (see Figures 2.1 and 2.2). These are namely the left and right windows, left, right, and center rearview mirrors as well as the front windshield, tachometer cluster, and the center stack (a.k.a. entertainment system column). Depending on the desired application of the gaze tracking system, these sectors may be changed according to the specific areas of interest in their respective applications. For this thesis, these nine sectors are used.

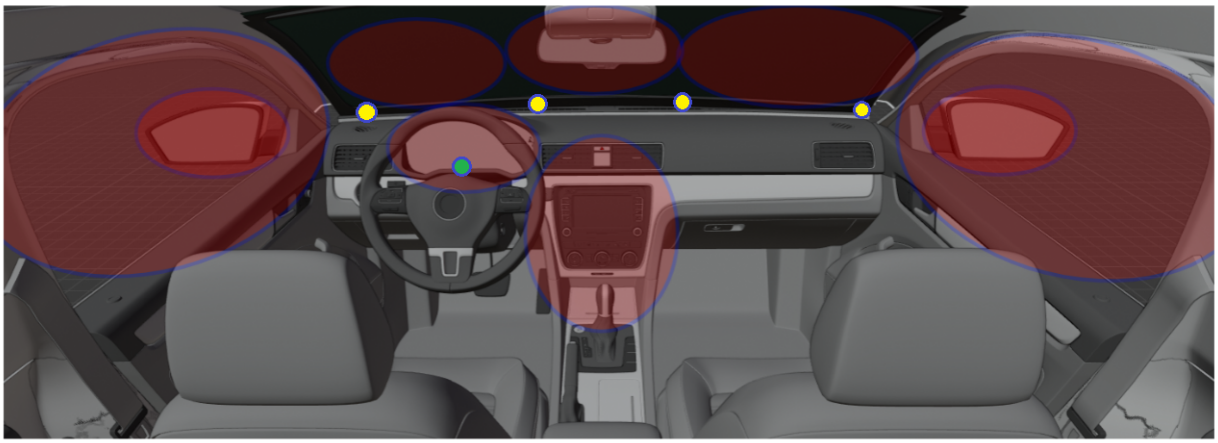


Figure 2.1: Rendering of the vehicle interior with the nine different sectors overlaid, marked with red ellipses. Placement of cameras is also shown, where the single-camera is marked in green and the multi-cameras are marked in yellow and green (five cameras). All cameras are aimed at the driver's face.

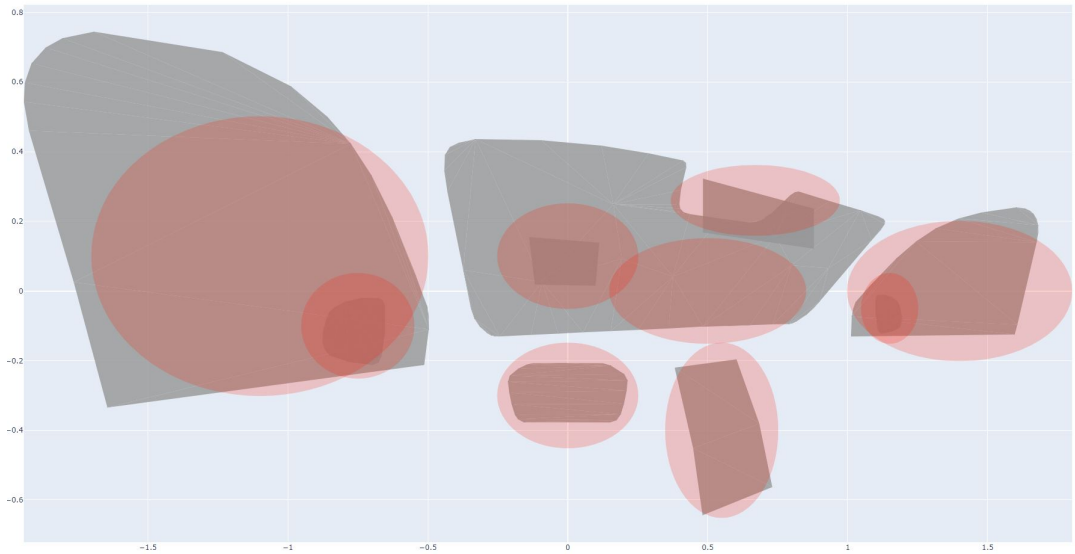
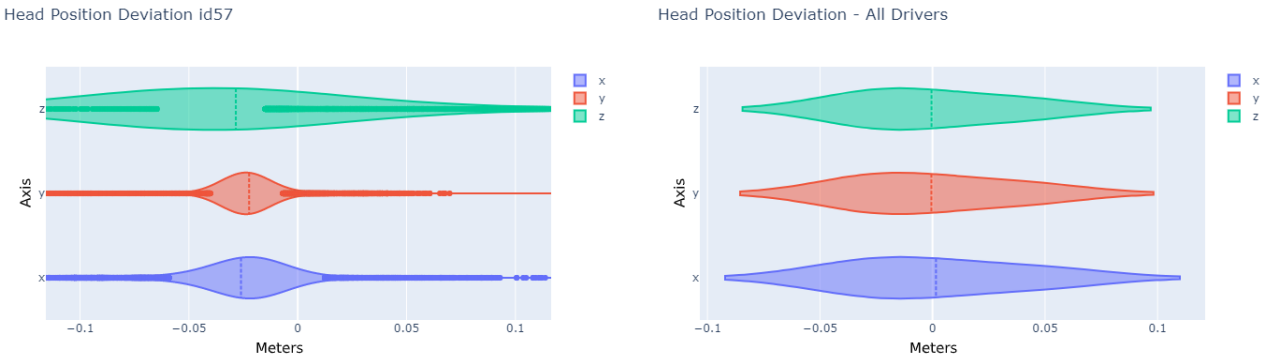


Figure 2.2: Projection of the nine sectors in the car, which corresponds to the red ellipses in Figure 2.1. Note that the perspective here is that of the driver.

2.1.2 Head Position

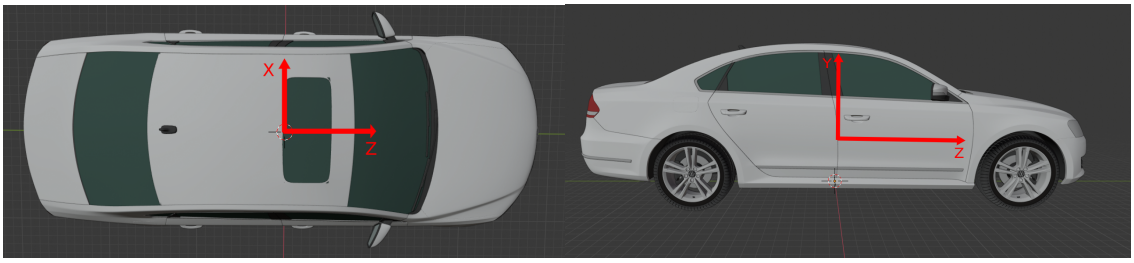
To be able to justify assumptions about the driver's head position during driving a statistical analysis had to take place. Both single individuals and groups of individuals were taken into account to be able to draw conclusions on this matter.

All available SEP data was used to generate this data, where the 3D head position of each individual from the data set was collected and plotted. These distributions, therefore, shows the average position of the head of the current driver, as well as the deviation amount from this point during driving. This data was visualized for one individual in subfigure (a) and for all individuals collectively in subfigure (b) of Figure 2.3. This was done to obtain information regarding the average deviation from the mean head position for an individual, as well as the span of locations where it is expected that the head positions of the majority of individuals lie within. The coordinate system is defined with the x-axis denoting the horizontal axis, increasing towards the left side of the car, the y-axis denoting the vertical axis, increasing towards the roof of the car, and the z-axis denoting the depth axis, increasing towards the front of the car as displayed in figure 2.4.



(a) One Individual (id57) **(b)** All Individuals

Figure 2.3: Violin plot of the deviation of head position for the driver during vehicle operation.



(a) Top view **(b)** Side view

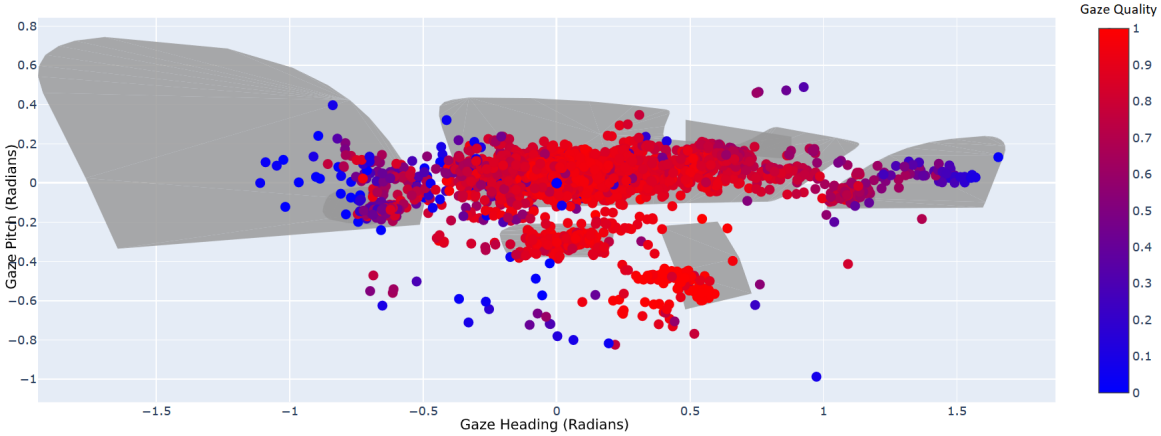
Figure 2.4: The coordinate system of the vehicle.

For the average person, the head movement during driving is typically quite small. One important difference between individuals is the mean head position, which is why the spread of the plot in (b) is wider and more populated than the plot of one individual in (a) of Figure 2.3. What is also notable in figure (a) is that in the x and y axis, the individual only tends to deviate from the mean head position with about 2.5 cm. It is only in the z axis the deviation is larger, but it is still only a deviation of about 10 cm. Altogether, this does not provide a notable change in the locations of the areas of interest from the perspective of the driver, either for one individual during a driving session or comparing different individuals to each other. This leads to the qualified conclusion that the circumstances for operation of the algorithms could be simplified as the head movement according to these tests is negligible.

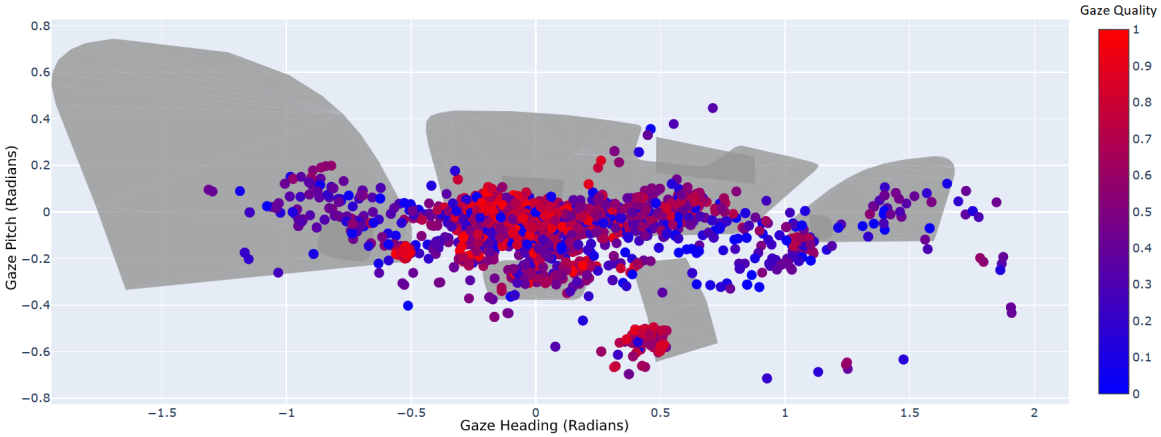
2.1.3 Track Quality

In addition to gauging the gaze direction and head direction, the Smart Eye algorithm also makes an assessment of the quality of these data points. For example, if the head is facing away such that only one eye is visible to the camera, then the quality of the gaze direction estimate will be affected and the algorithm gives this data point a lower grade in terms of quality. Although the quality of this data point is lower, it still contains information that can be used to make an assessment but it is helpful to know to what extent this can be trusted. Similarly, the head pose quality is also determined for each data point. The head direction quality was found to be sufficiently high consistently. The quality of data can be utilized to filter out data of substandard quality, as shown in Figure 2.5.

Further, to visualize how much of an impact the gaze quality has for each camera system (SEP and TC), an experiment was conducted. The two files were synchronized and for each time frame, the L2-norm between the corresponding SEP and TC point was stored. The hypothesis was that if the gaze quality is high for both the SEP dataframe and the TC dataframe, then the two gaze points are likely to be close to each other, while if the quality is low, then the measurement is less sure and the distance is likely to increase, as the measurement in the error should increase with the error in the measurement. This hypothesis was confirmed by the results shown in figure 2.6. Here, the average distance between SEP and TC gaze points are shown, and the smallest average difference is found when both the SEP quality and the TC quality is high.



(a) 15,000 samples of SEP data with corresponding quality



(b) 15,000 samples of TC data with corresponding quality

Figure 2.5: Gaze Heading (x-axis) and Gaze Pitch (y-axis) plotted on top of a 2D spherical projection of the car model around the mean head position of the driver. The data is captured from the same driving session. Note that the SEP data on average is of higher quality, and also that the TC data rarely is of high quality in the periphery.

TC Quality \ SEP Quality	0.0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0
0.0-0.1	0.12823	0.47596	0.43098	0.36361	0.33778	0.29652	0.26251	0.21620	0.17850	0.19979
0.1-0.2	0.74693	0.25918	0.25953	0.18883	0.19086	0.21104	0.22672	0.19715	0.16829	0.16700
0.2-0.3	0.78613	0.38303	0.24346	0.26982	0.20772	0.23981	0.23567	0.22417	0.18241	0.15366
0.3-0.4	1.01248	0.43226	0.37076	0.27183	0.24677	0.22769	0.20738	0.20661	0.17012	0.14790
0.4-0.5	1.22457	0.34869	0.24207	0.19982	0.19788	0.19449	0.19216	0.16889	0.14955	0.13850
0.5-0.6	0.68897	0.26468	0.18747	0.18491	0.15455	0.14529	0.15785	0.14606	0.13857	0.12757
0.6-0.7	0.53611	0.19132	0.23945	0.16896	0.14367	0.14614	0.15156	0.13918	0.12808	0.11734
0.7-0.8	0.48892	0.24126	0.28892	0.16459	0.16263	0.15236	0.15814	0.12820	0.12603	0.11826
0.8-0.9	0.46269	0.36316	0.25138	0.17344	0.17550	0.16049	0.15610	0.12290	0.12031	0.11404
0.9-1.0	0.44529	0.24693	0.25740	0.15839	0.14196	0.16919	0.14493	0.11366	0.11204	0.10620

Figure 2.6: Average distance between two synchronized points in a SEP Data file and TC Data file. Each data point is assigned a quality value between 0 and 1. Categorized by the combination of SEP gaze quality and TC gaze quality. Unit of the values in the cells is radians. Green color corresponds to smaller values, dark red to greater.

2.1.4 Gaze Cone

When calculating the gaze direction, the result is approximated as a single point described by a heading angle and a pitch angle originating from the eyes of the driver. However, in reality, humans do not focus on just a single spot, but rather on a circular area as projected from the eye. This area is referred to as the *gaze cone*. In the average person, the gaze cone makes up just less than 12° in diameter [14]. This means that even if the gaze direction could be perfectly estimated, it is likely that the actual area of focus is anywhere within the gaze cone, where the projected point is the center of the cone.

This gaze cone will be represented by a mean μ and a covariance $\Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$, where the mean is the actual gaze measurement for that data point and the standard deviation σ is the radius of the gaze cone in radians.

2.1.5 Input Data

To run the algorithms explained in chapter 2, the first step was to choose what data to use for clustering and gaze estimation. For the clustering itself, a natural approach for noting where the driver is currently looking is the gaze heading and gaze pitch. These measurements are the horizontal and vertical angles respectively for the direction of the gaze vector originating from the head of the driver. The reason for using these angles instead of the 3D gaze vectors themselves was mainly due to the very small amount of head movement that the driver shows during highway driving (displayed in section 2.1.2). This leads to no further accuracy by using a 3D

representation to describe each gaze point, and thus only provides more complexity to calculations and visualization.

For the ground truth generation however, the 3D gaze direction vectors were used to find the intersection objects for each measurement, as this was necessary to be able to interact with the model of the car and to provide the most accuracy possible.

A few different combinations of data were tested as input to the gaze estimation algorithm. The proposed idea is that there should be some correlation between head direction and gaze direction, as the head tends to move in the same direction as the gaze. Similar to the gaze direction, only the 2D euler angles head heading and head pitch were used to describe the head direction. In addition to this a head direction velocity ($\mathbf{v}_i$) was also calculated and applied in one of the versions, by dividing the difference from the current vector ($\mathbf{x}_i$) to the vector at the previous time step ($\mathbf{x}_{i-1}$) and dividing by the frame rate (R_f). This was done to see if this additional information would provide better resulting accuracy in the estimates.

$$\mathbf{v}_i = \frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{R_f} \tag{2.1}$$

2.2 Gaze Estimation

To ensure that a reasonable gaze sector estimate can be maintained even when the gaze tracking quality is lower, a relation between the head pose direction and the gaze direction needs to be established. The gaze tracking quality tends to decrease in the periphery towards the side windows and rear view mirrors due to the eyes facing away from the camera. In contrast, the head pose estimate remains accurate in these regions. Therefore, a gaze direction estimate based on the head pose direction could provide valuable data on what the driver might be directing their attention to. Further, such estimates also have the potential to provide more accuracy in cases where the gaze quality is high but the cluster probabilities are near a tie for two or more clusters.

This section introduces the two main methods tested in this thesis that aims to provide a solution to this described phenomenon. All methods are trained offline on SEP data as the accuracy and coverage of that system is greater than TC, and the trained model is then applied on TC data for online usage.

2.2.1 Correlation Grid

The Correlation Grid method was considered as a candidate to solve the gaze estimation problem during early discussions in the project. Using an input data set and a corresponding output data set, the method creates a grid of squares over the state space of the input data (see Figure 2.7) to divide it into subsections. This essentially becomes a 2D histogram, where every head pose measurement (input data) gets classified to belonging to a certain bin in the grid. During the training of

the grid, this is done for all data points, which leads to every measurement getting a corresponding bin.

The next step in the training is then to correlate these bins to corresponding gaze directions (output data), based on the output data for the measurements classified into the bins. To do this it is necessary to calculate the relative difference between gaze direction $\mathbf{y}_n$ (output data) and head pose direction $\mathbf{x}_n$ (input data) for each measurement n according to $\Delta_n = ||\mathbf{y}_n - \mathbf{x}_n||$. For each bin (i, j), the belonging Δ_n's are collected and used to calculate a covariance $\boldsymbol{\Sigma}_{i,j}$ and a mean $\boldsymbol{\mu}_{i,j}$.

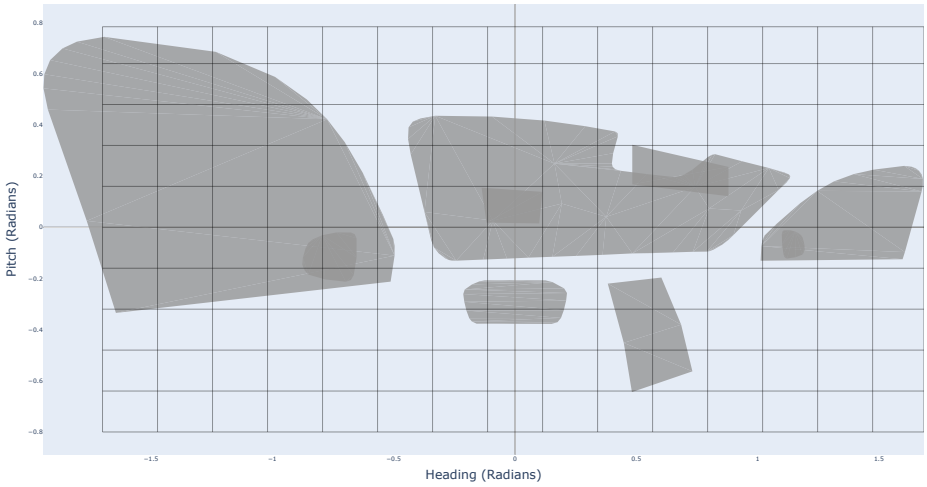


Figure 2.7: Grid with the width of 15 bins and height of 10 bins displayed on top of the projected interior of the vehicle from origo

To obtain the estimated gaze mean $\boldsymbol{\mu}_e$ and covariance $\boldsymbol{\Sigma}_e$ for a head pose measurement $\mathbf{x}_n$, its corresponding bin first gets identified with its grid coordinates i and j. The mean and covariance of the distribution for the measurement is then calculated according to equations 2.2 and 2.3, where $\boldsymbol{\mu}_{i,j}$ is the mean and $\boldsymbol{\Sigma}_{i,j}$ the covariance of bin i, j.

$$\boldsymbol{\mu}_e = \mathbf{x}_n + \boldsymbol{\mu}_{i,j} \tag{2.2}$$

$$\boldsymbol{\Sigma}_e = \boldsymbol{\Sigma}_{i,j} \tag{2.3}$$

The mean and covariance generated can then be used as a distribution to sample estimated gaze points from, which after having their respective cluster probabilities calculated can be summed and normalized to obtain a cluster probability vector representative of the distribution generated.

2.2.2 Regression Model

Due to the correlation grid having issues with total coverage over the entire state space, further research was done to find an alternative solution for comparison pur-

poses. An alternative was found in the Multi-layer Perceptron (MLP), which utilizes a simple neural network structure with hidden layers of neurons which, by introducing data sets for training adjusts the weights for each neuron to minimize the loss. This method can thus be used as a regression analysis tool to find a solution for a relationship problem between input and output data.

The methodology of the MLP is in its essence a simple neural network containing an input layer, one or more hidden layers, and an output layer. The neurons that these layers consist of are called perceptrons, and the idea of them was introduced in 1958 by Rosenblatt [15], but at that point, these perceptrons were only discussed as singular entities, however. As one perceptron alone just computes one output from many real-valued inputs through linear combinations using weights and a nonlinear activation function, it is not that useful on its own. However, when a number of these perceptrons are interconnected to obtain a neural network structure (see Figure 2.8) the result is the Multi-layer Perceptron model [16].

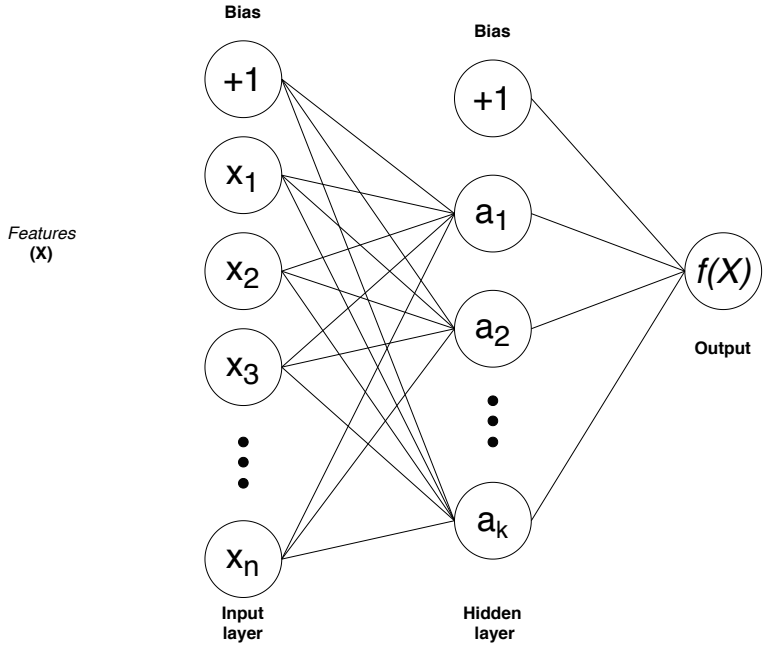


Figure 2.8: MLP model with one hidden layer.

This approach utilizes input data and a matching ground truth to train the network to predict the ground truth from the training data. This is done by adjusting the corresponding weights for each perceptron in the network to minimize the collective estimation error between the output of the network and the ground truth. This is done by regression, where the model trains using backpropagation without an activation function in the output layer. Because of this, the model uses the square error as the loss function. This was implemented using SciKit [17], where the foundation for the algorithm was already created.

Using a given set of training data $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_n, \mathbf{y}_n)$ where $\mathbf{x}_i \in \mathbf{R}^n$ is the input data vector for sample i and $\mathbf{y}_i \in \{0, 1\}$ the corresponding correct output.

An MLP with one hidden layer and one hidden neuron then learns the function described in equation 2.4.

$$f(x) = W_2(W_1^T\mathbf{x} + b_1) + b_2 \tag{2.4}$$

Here, $\mathbf{W_1} \in \mathbf{R}^m$ and $W_2, b_1, b_2 \in \mathbf{R}$ are model parameters. b_1 and b_2 represent the bias added to the hidden layer and output layer (displayed in Figure 2.8) and W_1, W_2 contains the weights of the input layer and hidden layer. The activation function is then given by the hyperbolic tan

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.5}$$

For a classification case that is not binary, i.e. with more than two classes, the learning function $f(x)$ (equation 2.4) becomes a vector of the same size as the number of the classes. Because of the regression functionality of the MLP model utilized in this thesis, the output simply becomes $f(x)$, where the output activation function just becomes the identity function.

A loss function is then applied with random initial weights, where the MLP is allowed to minimize this function by updating the weights and evaluating.

$$Loss(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{W}) = \frac{1}{2}||\hat{\mathbf{y}} - \mathbf{y}||_2^2 + \frac{\alpha}{2}||\mathbf{W}||_2^2 \tag{2.6}$$

Here, $\alpha||\mathbf{W}||_2^2$ is an L2-regularization term, which provides penalties onto complex models, and $\alpha > 0$ is just a non-negative parameter to adjust how much the penalty term affects the result. After this loss is calculated, a backward pass is used to propagate the loss onto the previous layers, which results in each weight parameter being given an update value to obtain a decrease in loss.

Given a large enough amount of data, this should then let the MLP regressor obtain a rough estimate of where the gaze point for a given measurement should be located given information about the head of the driver.

To not skew the estimate towards the more dense areas of the data, the input data needed to be scaled accordingly. This prevents faulty estimates due to the collection of data around for example the EOR-area (which receives about 62% of all gaze points, see 3.1), and gives the data set a more uniform distribution.

The MLP-model does not obtain a covariance estimate automatically based on the accuracy of the estimates, so this functionality needed to be implemented in another way. The solution became to exclude a few individuals from the training set, and then use this excluded data to obtain an accuracy measurement, and thus build a covariance from the difference between the estimate and ground truth of unseen data. With this addition, the method provides both a resulting estimated gaze mean $\boldsymbol{\mu}_e$ and covariance $\boldsymbol{\Sigma}_e$ for a head pose measurement $\mathbf{x}_n$. These values can later be used to sample estimated gaze points from, which after having their respective

cluster probabilities calculated can be summed and normalized to obtain a cluster probability vector representative of the distribution generated.

2.3 Clustering Methods

Clustering is a sub-method to one of the two fundamental parts of machine learning, namely unsupervised learning [18]. This method, contrary to supervised learning, does not require pre-annotated or labeled data to be able to extract information. Clustering algorithms are used to group data points into different classes, or clusters as they are also known. It is a technique that is highly utilized when the dimensionality of the data is higher than two or three, which is where the similarities between points become hard to distinguish by eye due to the difficulty in visualization. By applying a clustering algorithm on this type of problem, a more scientific approach on similarity statistics and correlation provides the necessary basis to draw a qualified conclusion from when no apparent correlation can be found by hand.

Despite this, clustering is used for low dimensional data in the means of this thesis, namely two dimensions (gaze direction in Euler angles). This is due to the need for self-classification and autonomy that is the purpose of the project, in which clustering provides a good and easily visualizable solution. In this section, the different clustering methods tested in the thesis are introduced and explained separately.

2.3.1 K-means

The k-means clustering method [19] was introduced in 1967. Each cluster contains one parameter; the cluster center represented as a coordinate in a d-dimensional space where d matches the number of dimensions of the data to be clustered. The algorithm aims to cluster n data points into k clusters by minimizing the sum of the squared distance between each data point and center of its assigned cluster. Each cluster j is defined by its center coordinate c_j, which is simply the mean value of that cluster, hence the name "k-means".

The algorithm is initialized by providing the number of clusters k and assigning random coordinate values to each of the k cluster centers $c_j, j = 1, ..., k$. Each data point x is then assigned to the cluster whose center point is closest to it. The notation $x_i^c = j$ refers to the ith point in the data set belonging to cluster j. The clusters are then assigned as such:

$$x_i^c = \arg_j \min(||x_i - c_j||, j = 1, ..., k) \tag{2.7}$$

The variable $m_{i,j}$ denotes if data point i is assigned to cluster j. If i is assigned to j, then $m_{i,j} = 1$, else $m_{i,j} = 0$. Once an assignment has been done, the mean position of all data points within one cluster is calculated and the cluster center is moved to this mean coordinate:

$$c_j = \frac{\sum_{i=1}^{n}(m_{i,j}x_i)}{\sum_{i=1}^{n} m_{i,j}} \tag{2.8}$$

Once all cluster centers have been updated, the algorithm repeats until no data points switch cluster label between two iterations; the algorithm is then said to have converged. However, it might also not be possible to find a converging solution. For such cases, a maximum number of iterations is determined and the algorithm halts once this number of iterations has been reached. Since the number of clusters k is given as input to the algorithm, the user needs to know how many clusters can be expected to find in the data set.

This algorithm is easy to implement and runs fast, however, it works best when the clusters are assumed to be spherical. This leads to difficulties when more complex cluster shapes are required for the implementation, which is the case in this thesis and is therefore not ideal for all problems. Its simplicity also means that the algorithm is less suitable for more complex data sets, such as the ones used for this project. Because of this, the K-means algorithm was not decided to be a further candidate for this thesis.

2.3.2 Fuzzy C-means

The Fuzzy C-means algorithm was presented and developed by Dunn in 1973 [20], and an improved version was later introduced by Bezdek in 1984 [21]. Fuzzy C-means can be regarded as an extension to the K-means algorithm (see section 2.3.1) with the difference that the cluster labeling of each point is non-binary. Instead, each data of the N points x_i is assigned a floating-point value between 0 and 1 for each cluster. In other words, each data point can belong to several clusters simultaneously, hence the "fuzzy" part of the name.

The algorithm is initialized by specifying the number of clusters the model contains (the "C" parameter), as well as selecting random initial values for the cluster belonging of each data point in the "U" matrix. This matrix consists of elements $u_{i,j}$ that corresponds to the probability that data point i belongs to cluster j. The center coordinate c_j of each cluster j is then calculated as such:

$$c_j = \frac{\sum_{i=1}^{N} u_{i,j}^m x_i}{\sum_{i=1}^{N} u_{i,j}^m} \tag{2.9}$$

The next weight matrix $U^{(k+1)}$ is then updated as such:

$$u_{i,j} = \frac{1}{\sum_{k=1}^{C} \left(\frac{||x_i - c_j||}{||x_i - c_k||}\right)^{\frac{2}{m-1}}} \tag{2.10}$$

If the norm difference between the new and the previous weight matrices exceeds some parameters ϵ, then the algorithm has converged:

$$||U^{(k+1)} - U^{(k)}|| < \epsilon \tag{2.11}$$

Else, the algorithm resumes at the center coordinate calculation step.

As with the K-means algorithm (see section 2.3.1), Fuzzy C-means is easy to implement and runs fast on the data available in this project. The addition of the fuzzy logic provides valuable additional information regarding the probabilities of the data points belonging to all clusters, which is a requirement in this project. Regardless, the clustering ability of this algorithm is biased towards creating simple, equidistant clusters which are not useful for this project. Hence it was not considered as a further candidate for this thesis.

2.3.3 Expectation Maximization

Expectation-Maximization is a clustering method utilized in the so-called *Gaussian Mixture Model* type algorithms. Each cluster is defined as a k-dimensional Gaussian approximation with the parameters mean and covariances. The parameters for each cluster are updated through maximum likelihood estimates until convergence, which is defined as a change in parameters lower than a set tolerance.

As explained in [22] Expectation-Maximization utilizes two major steps in its algorithm. First, the expectation step, or *E-step*, is carried out, which utilizes either the randomized or the pre-defined means and covariances for the clusters and calculates the relative probabilities for each point in the introduced dataset belonging to each of the clusters. For each point, all of its probabilities are normalized to obtain a full probability of each point belonging to each cluster.

As each cluster is defined as being a Gaussian distribution, $\mathcal{N}(\mu, \Sigma)$ can be used to describe every cluster. These clusters are initialized randomly though this equation. It is then possible to describe the probability of a point $\mathbf{x}_n$ belonging to cluster k by the following equation.

$$p(z_{nk} = 1|\mathbf{x}_n) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\mu_j, \Sigma_j)} = \gamma(z_{nk}) \tag{2.12}$$

Where z is a binary variable that becomes 1 when x_n belongs to cluster k and 0 otherwise. The *E-step* equation then boils down to simply solving this equation using the old parameter values.

After this, the maximization step, or *M-step*, commences. The maximization is calculated and evaluated by a log-likelihood function until convergence, which then yields the new means and covariances for all clusters based on the data fitted.

The first part of the *M-step* creates a maximizable function to utilize in optimizing the best placements and sizes for the clusters. This equation is defined as

$$\mathcal{Q}(\theta^*, \theta) = \mathbb{E}[\ln p(\mathbf{X}, \mathbf{Z}|\theta^*)] = \sum_{\mathbf{Z}} p(\mathbf{Z}, \mathbf{X}|\theta) \ln p(\mathbf{X}, \mathbf{Z}|\theta^*) \tag{2.13}$$

Here equation 2.12 substitutes $p(\mathbf{Z}, \mathbf{X}|\theta)$, which together with $\ln p(\mathbf{X}, \mathbf{Z}|\theta^*)$ being defined as

$$\ln p(\mathbf{X}, \mathbf{Z}|\theta^*) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk}[\ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)] \tag{2.14}$$

Which is the logarithmic expression of the joint probability calculation of all observations and z's. Putting equations 2.12 and 2.14 into equation 2.13 thus results in the final equation as

$$\mathcal{Q}(\theta^*, \theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk})[\ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)] \tag{2.15}$$

Where $\theta = \{\pi, \mu, \Sigma\}$ is the model parameters and θ^* is the revised parameters needed to be found through maximizing equation 2.15.

These revised parameters are found using $\theta^* = argmax_\theta \mathcal{Q}(\theta^*, \theta)$, but $\mathcal{Q}$ also needs to incorporate the fact that all π-values needs to sum up to one. This is done through the addition of a Lagrange multiplier to equation 2.15 which results in the following equation.

$$\mathcal{Q}(\theta^*, \theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk})[\ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)] - \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right) \tag{2.16}$$

Now it is possible to determine the parameters using maximum likelihood. This is done by first taking the derivative $\frac{\partial \mathcal{Q}(\theta^*, \theta)}{\partial \pi_k}$ and setting it equal to zero. Then this equation is rearranged to obtain an expression for π_k, and using the previously stated fact about the probabilities over k summing to 1 yields

$$\pi_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk})}{N} \tag{2.17}$$

In the same manner, the other two derivatives $\frac{\partial \mathcal{Q}(\theta^*, \theta)}{\partial \mu_k}$ and $\frac{\partial \mathcal{Q}(\theta^*, \theta)}{\partial \Sigma_k}$ are calculated and rearranged with the same assumptions to obtain the last two equations used in the *M-step* as follows.

$$\mu_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})} \tag{2.18}$$

$$\Sigma_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})} \tag{2.19}$$

These are then iterated until the likelihood value calculated in equation 2.20 converges to a local maximum.

$$\ln p(\mathbf{X}) = \ln \prod_{n=1}^{N} p(\mathbf{x}_n) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k) \tag{2.20}$$

This algorithm was first mentioned in 1977 in [23], and one of the earliest applications was for counting genes to estimate allele frequencies. The seemingly simple

approach opened up big opportunities to analyze and classify data of low dimensionality. It is a very intuitive solution which makes the ease of use and the application of the algorithm much easier compared to most of the recent approaches to the clustering problem.

This clustering method was despite its seemingly complex structure quite fast to run, but consists of more computations than the simpler algorithms like K-means (section 2.3.1) or Fuzzy C-means (section 2.3.2). This is nothing that impacts the real-time usage in any noticeable way, however, and with the perks of the cluster definitions and update scheme, this is one of the clustering algorithms that seemed promising for future use in the means of this project.

2.3.4 Gaussian Binary Gravity (Modified EM)

The standard Expectation-Maximization algorithm brings a lot of useful concepts and approaches to the clustering problem faced in this thesis, for example, the definition of the clusters as multivariate gaussian distributions. Some of its inner workings, however, are not optimal when data cannot be assumed to be normalized and evenly distributed. It is also not ideal when the placements and distributions of the clusters can be assumed to have logical placement and size based on a predetermined model to which the data needs to be correlated, which is the case for this thesis. Due to the problem being of this character, a new modified version loosely based on the Expectation-Maximization algorithm is proposed to utilize the intuitive character of the data and the sought after result.

There are two main differences between this new proposed algorithm and the original EM algorithm. Firstly, as the areas that are of interest of being detected have a known and fixed size, the covariances $\boldsymbol{\Sigma}_k$ are fixed when initializing the algorithm and does not change during the maximization step. This provides the proper coverage for each area of interest by its intended cluster and removes the possibility of the cluster shrinking if the data provided in that area would be more compact than expected. If this was not the case, and new data not being as close to the first datum but still within the area of interest was introduced, the new data would have a higher risk of being misclassified. This would also destroy the correlation between the predetermined model and the clusters.

Secondly, the means of each cluster $\boldsymbol{\mu}_{i,k}$, which are the values actively updated during the algorithm, are only affected by the points which have the largest individual probability of belonging to said cluster. This is done through weighting each of the points being most probable to belong to a certain cluster based on the percentage of their probability belonging to the cluster. This then scales the vector between the current cluster center and the point itself, which is done for all points belonging to the cluster and is then normalized by dividing with the total number of points in the dataset. This provides robustness to the absence of data points in certain areas where it is known there exists a cluster, but the driver has simply not gazed in that area yet.

Furthermore, this approach does not require to pre-train the algorithm with any data as it is designed to adjust for each individual separately and only the initialization dictates what accuracy can be gained and what the model should incorporate.

The prediction step simply uses equation 2.21 to calculate the probability vector of measurement $m = 1, ..., M$ belonging to each of the clusters k at iteration i. Here $\boldsymbol{\mu}_{i,k}$ denotes the center and $\boldsymbol{\Sigma}_k$ the covariance of cluster k at iteration i. As the covariance never changes no iteration index is needed on that variable. Further, K symbolizes the total amount of clusters.

$$\mathbf{p}_{i,m} = \frac{\sum_{k=1}^{K} \mathcal{N}(\mathbf{x}_m | \boldsymbol{\mu}_{i,k}, \boldsymbol{\Sigma}_k)}{K} \tag{2.21}$$

The update step then uses this prediction together with the binary variable $z_{i,m,k}$ (see equation 2.22) to locate which cluster the measurement has the highest probability of belonging to, given the current cluster locations. The vector between the measurement and the inherent cluster is then combined with the binary variable, as well as the probability of the measurement belonging to that cluster and the gaze quality for the measurement. This is done for all points and divided on the number of points for each cluster, to obtain one moving vector $\mathbf{v}_{i,k}$ for each cluster k in iteration i as can be seen in equation 2.23. The cluster center locations are then updated through equation 2.24 and the whole process iterates until $\boldsymbol{\mu}_{i+1,k} - \boldsymbol{\mu}_{i,k} < tol$. Here, M symbolizes the total amount of measurements.

$$z_{i,m,k} = \begin{cases} 1 & \text{if } argmax(\mathbf{p}_{i,m}) = k \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

$$\mathbf{v}_{i,k} = \frac{\sum_{m=1}^{M} (\mathbf{x}_m - \boldsymbol{\mu}_{i,k}) z_{i,m,k} p_{i,m,k} Q_m}{M} \tag{2.23}$$

$$\boldsymbol{\mu}_{i+1,k} = \boldsymbol{\mu}_{i,k} + \mathbf{v}_{i,k} \tag{2.24}$$

When it comes to suitability in the realm of this project, the GBG algorithm has the most upside of all clustering methods that has been introduced. The structure of the update step creates a good basis for maintaining the model integrity of the areas of interest and makes it a prime candidate for the potential candidate for this thesis, despite the calculation strain being slightly higher than any other method in this chapter.

2.3.5 Density-Based Spatial Clustering of Applications with Noise

Density-Based Spatial Clustering of Applications with Noise, or *DBSCAN*, was first proposed in 1996 [24]. The algorithm takes the dataset and two parameters, the neighborhood radius ϵ and the neighbor amount threshold d_{min}, as input. For each

unlabelled data point x_i in the input, an n-dimensional sphere neighborhood of radius ϵ is created. The number of points inside this neighborhood is calculated, and if this number, including the datapoint x_i, is lower than d_{min} then x_i is labeled as noise. Else, x_i is classified as a core point and the set of remaining points within the neighborhood are classified as border points within the same cluster. If any of the neighboring points belong to a different cluster already, all of these points are updated so that they belong to the same cluster. The algorithm halts once there are no unlabelled data points left. The algorithm is deterministic in the sense that it will always reach the same cluster labeling as output given that the same input data and the same parameters were used.

With appropriate parameter values, the algorithm proved quite successful in finding and separating major objects, such as eyes on road (EOR), front windshield, or the tachometer cluster, which was relevant to this project. However, the algorithm was not successful in separating "clusters in clusters", such as the left side mirror which, when projected from the driver's head position lies within the left window. Further, the clustering is computationally heavy and is therefore not an ideal candidate for a system that needs to be able to run online to continuously update the clusters during use.

2.3.6 Density-Based Clustering with Constraints

Density-Based clustering with constraints [25], or *CDBSCAN*, was proposed as an extension of DBSCAN. By providing constraints in the form of *cannot-link* and *must-link*, data points are forced to belong to different clusters or merged to belong to the same cluster respectively. The authors of [25] present one of the purposes of this algorithm to be to cluster geographical data within the field of cartography where there might be a priori known barriers, such as rivers. By designing corresponding cannot-link constraints for points that are connected over the river would force clusters spanning across a river to be split up.

The original idea in this project was to first attempt to label data offline, then utilize this pre-labelled data to train a model which then could be used to define cluster centers, between which cannot-link constraints could be defined. Similarly, data points very close to these cluster centers could be forced to join that cluster by imposing must-link constraints. However, although the data in this project can be modelled as a geometrical data, the input data is subject to measurement errors that resulted in inaccuracies in terms of assessing where the cluster centers actually should be as opposed to where they are calculated to be. If the exact positions of cluster centers were known, then there would be no reason to cluster the data. Therefore, this method has been discarded due to difficulties in defining accurate constraints.

2.4 Gaze Sector Model Correction

The theory studied so far in this chapter can be applied without looking at temporal data. The clustering methods, for example, look at a given data point but draw no conclusion from what cluster the previous data point in time was assigned. It is assumed, however, that useful information could be extracted by analyzing patterns in data points over time. One hypothesis is that a driver might, in an attempt to check the environment behind the car, go from the left rear mirror to the right rear mirror, i.e. show a higher probability of going to the right rear mirror if the previous sector focused on was the left rear mirror. This sub-chapter outlines the Markov chain, and how this can be used to increase the accuracy in the prediction of low-quality data points.

2.4.1 Markov Chain

A Markov Chain is a model used within system theory. The Markov Chain contains two properties: *states* and *events*, see Figure 2.9. In terms of this project, states are used to represent the sectors of the car, e.g. eyes on road, central rear mirror, left side mirror, while events represent the transition between states that happens between two adjacent timeframes. Note that the set of transitions include self-loops; a transition can start and end in the same state. The model describes the probability of events transitioning between the states. The model is created from data where a sequence of state-event-state is known. Based on this, it is possible to calculate an a priori estimate of the probability that each event can transition to any other state (or itself). Once the model has been trained, the process is memoryless. This means that the process only looks at the current state while ignoring the previously visited states. The hidden Markov model described in chapter 2.4.2 is an extension of the Markov chain.

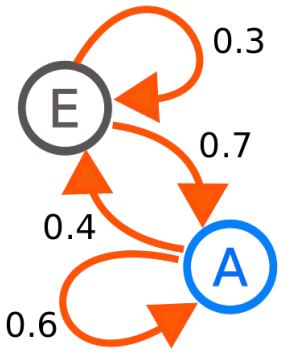


Figure 2.9: Markov Chain with two states A and E. There are two possible events at each state: switch state or remain. The probability of each event happening is represented by the number next to each event's arrow.

2.4.2 Hidden Markov Model

The hidden Markov model is an extension of the Markov chain. The system is assumed to contain two sets of states: observable and non-observable (hidden) states. The model is constructed and trained from an input consisting of one sequence of states for each set. Based on these two sequences, the idea is to analyze the probability that a transition occurs in the hidden states as well as the probability that an observable state coincides with a hidden state. Three assumptions are made:

(i) At any given time, the system visits one and only one of the hidden states, and at each time one and only one state is observed.

(ii) Each of these sets can be modeled as a Markov chain, a model can be built to predict the probability of the hidden states given the observable states.

(iii) The two sequences align temporally, such that each item in the hidden sequence has a corresponding item in the observed sequence that occurred at the same time.

Let us call the set of hidden states $X = \{x_i\}, i = 1, ..., n$ where n is the number of hidden states, and the set of observable states $Y = \{y_i\}, i = 1, ..., m$ where m is the number of observable states. The input is a sequence of hidden states $\{h_i\} : h_i \in X \ \forall i$ of length n_h, and analogously a sequence of observable states $\{o_i\} : o_i \in X \ \forall i$ of length n_o. The hidden Markov model consists of three matrices that need to be computed using the input above: *initial probability matrix*, *transition probability matrix*, and *emission probability matrix*.

The initial probability is simply the occurrence frequency of each of the hidden states. This gives $\{\frac{k_i}{\sum_i^n k_i}\}$, $i = 1, ..., n$ where k_i denotes the number of occurrences of hidden state x_i in the hidden sequence. Note that the sum of this vector is one.

The transition probability matrix is the Markov chain for hidden states, as explained in Chapter 2.4.1. This provides a matrix such that element i, j of the matrix corresponds to the probability that each hidden state $x_i \in X$ transitions to state x_j. Note that x_i and x_j may refer to the same state, such that $x_i = x_j$ if the transition returns to the same state. The rows are normalized such that the sum of each row equals one, and the result is thus a square matrix of probabilities of size $n_h \times n_o$.

The emission probability represents the probability that an observed state y corresponds to a hidden state x at a given point in time t. The emission probability matrix is calculated by iterating over the sequence of hidden state and the sequence of observed states simultaneously. For each (synchronized) time step, the correlation between hidden state x_i and observed state y_j is stored by increasing the value of element i, j in the matrix by one. Once the sequence is complete, each row is normalized. This matrix is rectangular of shape $n_h \times n_o$. The emission probability matrix is the very essence of the hidden Markov model as it answers the following question: given an observable state y, what actual (hidden) state x does this correspond to?

The hidden Markov model is necessary to use the Viterbi algorithm described in

chapter 2.4.3.

2.4.3 Viterbi Algorithm

The *Viterbi algorithm* [26] is an algorithm that can be used to find the most likely sequence of hidden states in a hidden Markov model, given a sequence of observed states. The prerequisites are the three matrices outlined in chapter 2.4.2. The algorithm uses dynamic programming to find a path that corresponds to the sequence of states with the highest likelihood based on the sequence of observed states.

Given a problem with n_h hidden states and n_o observable states, the following is obtained from the hidden Markov model:

- initial probability matrix P of size $n_h \times 1$
- transition probability matrix T of size $n_h \times n_h$
- emission probability matrix E of size $n_h \times n_o$

The algorithm also requires a sequence of observed states, $\{o_i\}$ of size n_e to be evaluated.

The algorithm keeps track of two tables T_1 and T_2 of size $n_h \times n_e$ (the number of hidden states multiplied by the length of the observed sequence to be evaluated). Here index i, j of T_1 stores the probability of the most likely path of hidden states $x_1, ..., x_k$ where x_k is the ith element in the most likely hidden sequence, given the observed sequence $y_1, ..., y_j$. The table T_2 is the corresponding most likely path of table T_1.

The algorithm is initialized by populating the first row of the T_1 with the multiplication product of the starting probability and the corresponding likelihood of the first item in the observed sequence from the emission probability matrix. The T_1 table is then populated for each element in the observed sequence such that the most likely path from the following index as obtained from the product sum of the emission probability matrix and the transition probability matrix and the corresponding item in the previous index of the T_1 table. Similarly, that index is then stored in the T_2 table. When this has been completed for all elements in the observed sequence, the most likely path is found by backtracing the T_2, starting in the element which had the highest probability at the last index.

The output of the Viterbi algorithm is a sequence of hidden states such that the likelihood is maximized. This sequence can be visualized in a *Trellis diagram*, see Figure 2.10.

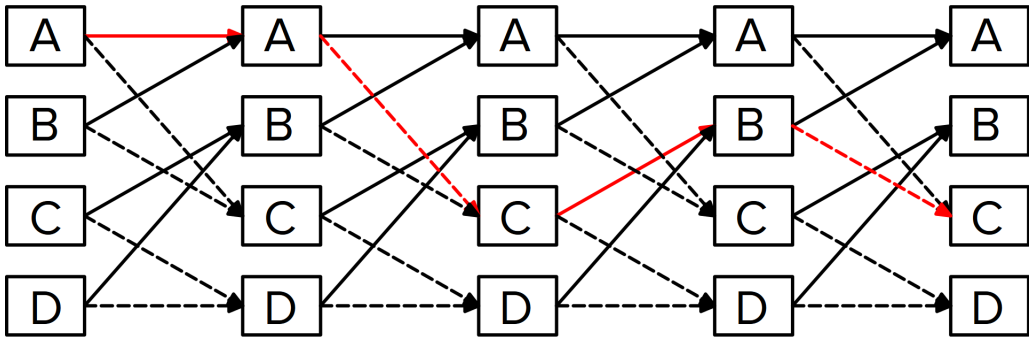


Figure 2.10: Example of a Trellis Diagram showing the most likely path (red) between four hidden states (A, B, C, D).

The output of the Viterbi algorithm (further explained in chapter 3.4) did not improve the overall accuracy of the algorithm and was thus discarded. However, further research is encouraged, specifically in terms of pre-processing the input data to the hidden Markov model.

3
Results

This chapter presents and discusses the results obtained during this project. Section 3.1 looks at methods of estimating the gaze when the gaze quality is low. Results from experiments showed that a multi-layer perceptron regressor provided the most convincing results, and this method was thus chosen to estimate gaze direction for the final algorithm.

Next, in section 3.2, the proposed clustering methods are evaluated. The gaussian binary gravity clustering method was chosen as the most suitable method due to this method providing accurate clustering results while also being simple to initialize and update.

Then, in section 3.3 the results from the cluster update step are shown. This is an essential step in providing an automatic calibration of the algorithm while it is running online.

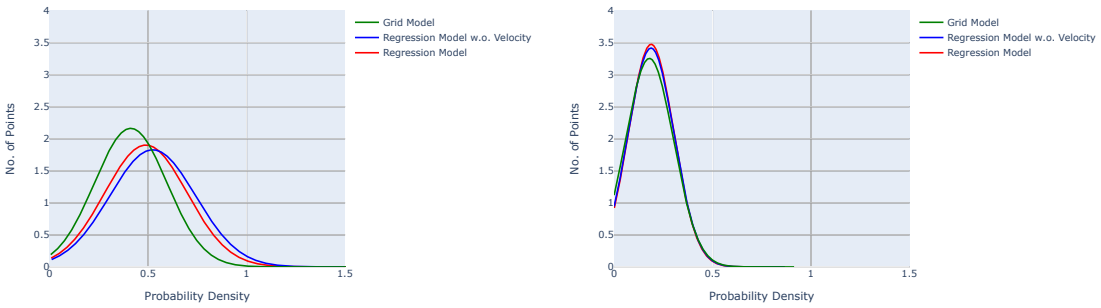
Lastly, in section 3.4, results are shown from the proposed method to perform statistical correction using the Viterbi algorithm on a pre-trained hidden Markov model. The results from this algorithm did not improve the accuracy of the final algorithm and were thus discarded from use there. However, as this method was deemed promising for further work, the results are still displayed.

3.1 Gaze Estimation

This section presents and discusses the obtained results for the approaches and methods tested in this project related to the manipulation of raw data. Specifically, the objective of these methods is to utilize information from the head pose direction to estimate a more accurate gaze direction when the gaze quality is low. This will in turn improve accuracy in the classification as well as the cluster update step.

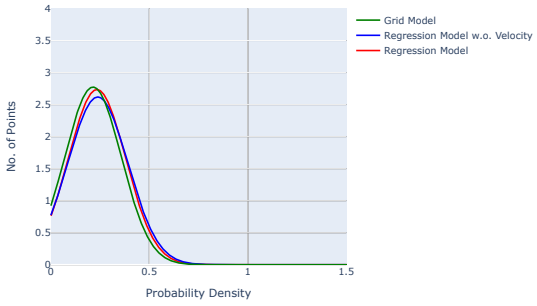
The main comparison tool used in this section is the distribution of the probability densities for the estimates, such as the ones shown in Figure 3.1. Given an input (such as the head position), an estimator is given as an output with a mean position and a covariance. This estimator is denoted $\mathcal{N}(\mathbf{x}_{true}|\boldsymbol{\mu}_{est}, \boldsymbol{\Sigma}_{est})$. The closer to the center a proposed "true" gaze point $\mathbf{x}_{true}$ is, the greater the value of the output of the estimate is. These densities are calculated for a data set, and the distribution of these densities is what can be seen in Figure 3.1. Also, since the integral of the

probability density distribution is always 1, the maximum value of the density of the function increases as the covariance decreases. As the shape of the probability density function for a given input comes from the data it has been trained on, a more confident estimate will have a smaller covariance and a less confident estimate will have a larger one. This will result in higher density values for confident estimates being reasonably correct, but much lower density values when the estimates are incorrect. The less confident estimates do not become as penalized for having an imperfect estimate but do not gain as high of a density value when the estimate is good, as the covariance is quite large.



(a) $Q_{gaze} < 0.5$

(b) $Q_{gaze} > 0.8$



(c) All Q_{gaze}

Figure 3.1: Distributions of probability densities from estimates for each of the three implementations of gaze estimators

In Figure 3.1, three implementations are compared. Firstly the Correlation Grid (section 2.2.1) is displayed in green, and secondly two different versions of the regression model (section 2.2.2); one using only head position to estimate the gaze direction in red, and one using both head position and velocity to estimate the gaze direction in blue, are included. Subfigure (a) shows the performance in the situations where the gaze quality is considered bad ($Q_{gaze} < 0.5$, subfigure (b) shows the performance when the gaze quality is considered good ($Q_{gaze} > 0.8$) and subfigure (c) displays the collective performance for all gaze qualities.

3.1.1 Correlation Grid

To obtain results that could be evaluated and compared, a few different parameters of the grid itself were adjusted. The adjustable parameters used were the grid size itself, the contribution of each point to its respective bin, as well as the number of files used to train the grid. All training was performed using data from the multi-camera system (SEP) to ensure the best accuracy of the measurements as possible for offline use.

First the grid size was evaluated between having the bin-resolution of 15x10 (see Figure 2.7) and 30x20 (see Figure 3.2). It is deductible even from simply evaluating the grids by eye that in the latter case, the bins are very small compared to the corresponding projected sectors of the vehicle. This subsequently leads to the risk of the head pose measurements belonging to a certain bin being few and thus not having enough data to provide a reliable distribution. Because the main use for this grid is to obtain a rough estimate to help deduct in which general area the current gaze point should reside in, this will lead to a bad estimation in each bin, as not enough data exists to build a dependable distribution from. The differences between neighboring bins also tend to be larger because of this, especially closer to the periphery. Because of this, it was decided to use the bin resolution of 15x10 when implementing this method.

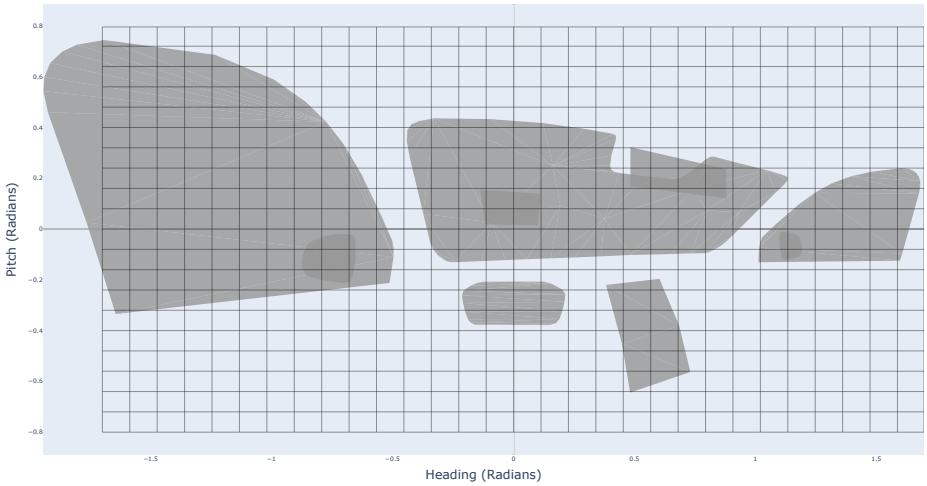


Figure 3.2: Grid with the width of 30 bins and height of 20 bins displayed on top of the projected interior of the vehicle from origo

In the aspect of the contribution every point has on its corresponding bin, a couple of different methods were used. The first was to simply discard all points with a $Q_{gaze} < 0.9$ from making any impact at all. This yielded a robust estimation, but the coverage over the entire grid became significantly worse. This is especially unwanted in this circumstance as the main use of this method is to get estimates where the Q_{gaze} tends to be sub-optimal. Because of this, the other method implemented a simple linear function (Equation 3.1) depending on the Q_{gaze} of the measurement.

This equation results in a gaze multiplier M_{gaze} that is used to scale each contribution more suitably according to the corresponding gaze quality.

This gaze multiplier is used during training to penalize the contributing samples of lesser quality and inhibit them from unresonably diverging the estimate. Each contribution to a bin for one measurement was named $\mathbf{z}$, and the calculation is described in equation 3.2. Here $\mathbf{x}$ denotes the head pose for one measurement and $\mathbf{y}$ the gaze direction for the same measurement.

$$M_{gaze} = \begin{cases} 1 & \text{if } Q_{gaze} > T_M \\ \frac{1}{T_M - T_m} Q_{gaze} + 1 - \frac{T_M}{T_M - T_m} & \text{if } T_M > Q_{gaze} < T_m \\ 0 & \text{if } T_m > Q_{gaze} \end{cases} \tag{3.1}$$

$$\mathbf{z} = (\mathbf{y} - \mathbf{x}) M_{gaze} \tag{3.2}$$

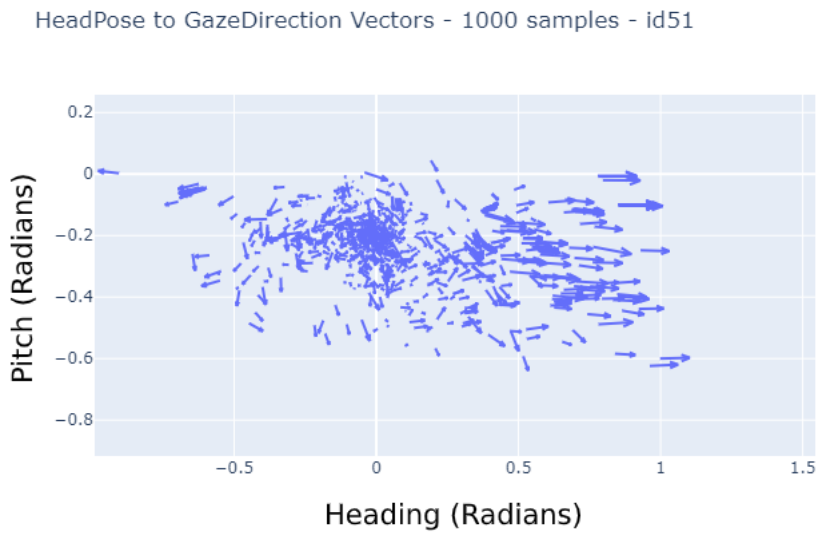


Figure 3.3: Vector plot with 1000 samples going from head pose direction to gaze direction for a single individual.

What became increasingly prevalent during testing was that the correlation between head direction and gaze direction had quite the opposite result of the hypothesis of the behavior that was based on early tests that displayed raw samples from the data (see Figure 3.3). Instead of showing results of gaze points located further towards the periphery when the head direction deviates from the immediate, straight-forward direction, the collective gaze directions tended to be directed towards the center as can be seen in Figures 3.4 and 3.5. After some discussion, it was deducted that this was the result of neglecting the velocity vector from the training of the grid, which with its absence did not account for the moments when the intended gaze point is moving and not fixed on an object. This led to the discovery of an interesting phenomenon in which the driver tends to lead the movement of the gaze direction

with the eyes, and then the head follows a split second later. This leads to some gaze points that are already moving being registered in the bin which the current head direction is residing in, before the head movement commences and follows the eyes. Subsequently, due to the sheer amount of data points consisting of gaze points in the general EOR (eyes on road) area and that this area is the most common one to switch to, this led to a bias being created towards the center of the coordinate system.

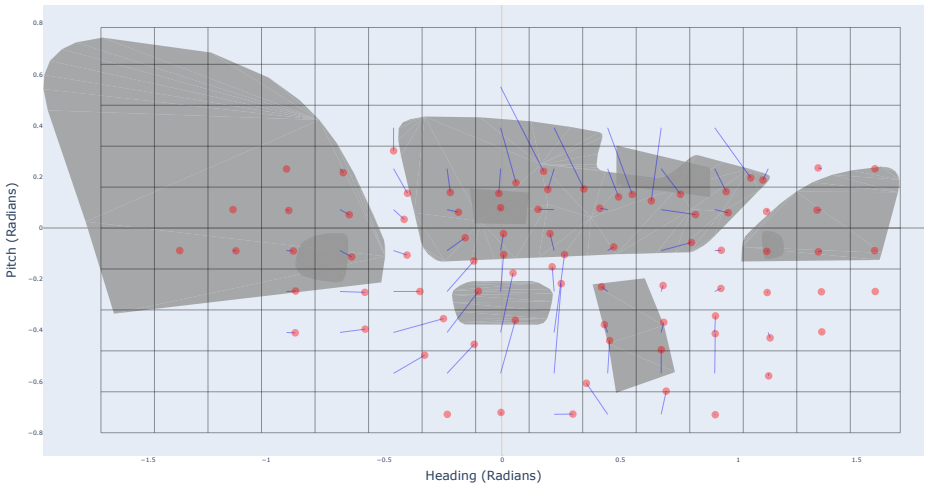


Figure 3.4: 15-10 Grid trained on 10 individuals with a linear quality impact ratio. The red dots represent the corresponding mean of the gaze distribution of the bin from which the blue line originates from.

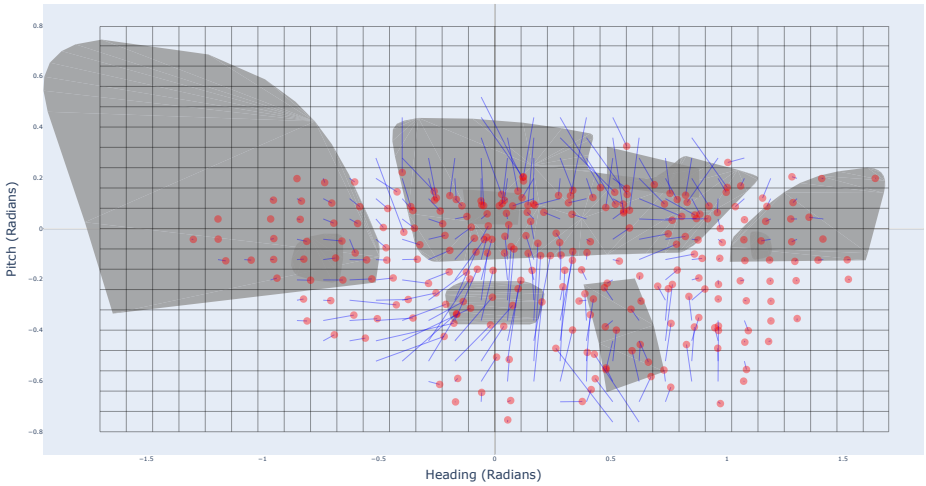


Figure 3.5: 30-20 Grid trained on 10 individuals with a linear quality impact ratio. The red dots represent the corresponding mean of the gaze distribution of the bin from which the blue line originates from.

The results that this method therefore yielded could provide helpful data about the whereabouts of the gaze direction based on the head direction, but is far from perfect. However, it provided some usable knowledge about the gaze tendencies during head movement that could provide helpful insights in the future.

3.1.2 Regression model

The Multi-layer Perceptron Regressor in general obtained quite reasonable accuracy results in general. This can be noted from Figure 3.6 where the average difference between the predicted gaze direction and the true gaze direction is displayed depending on how many individuals were used for training the model. From this figure, it is notable that regardless of how many individuals are used for training the model, the average distance between the estimated gaze direction and the true one is circa 0.2 radians (12 degrees). This can be compared to the general uncertainty for a real gaze measurement based on the gaze cone introduced in section 2.1.4, which has a radius of about 0.1 radians. To obtain only a slightly worse accuracy between the estimate and the true measurement compared to the accuracy of the actual gaze measurements is therefore promising.

However, the output itself simply generates a predicted absolute position of the gaze direction. Due to the prediction not being perfect and to the structure of the rest of the system, this will yield results that deviate from the true gaze direction in the majority of the time and it will not be possible to interpolate between the predictions and the measurements in a probabilistic manner. Therefore it was decided that the results of these tests can be used to create an uncertainty covariance around the predicted point to obtain a probability distribution of where the true gaze measurement should be located. By doing this, and also defining the true gaze point as a probability distribution based on the gaze cone (see section 2.1.4), it enables an intersection of the covariances to be extracted and thus creating a weighted estimate based both on the estimate and the true gaze direction.

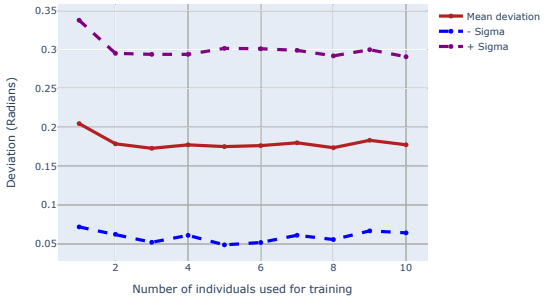
The distribution was created through training the covariance of the already trained MLP model with new data sets, where the covariance around the estimate became the variance of the estimates compared to the true gaze points for these measurements. This also enables the testing and comparison between this method and the Grid Correlation to be carried out in a more just way. This procedure generated the following covariance matrix for the model using only head pose direction:

$$\mathbf{\Sigma}_{est} = \begin{bmatrix} 0.0192 & -0.0058 \\ -0.0058 & 0.0202 \end{bmatrix}$$

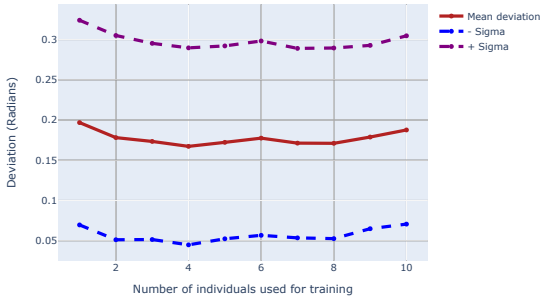
And the following covariance matrix for the model using both head pose direction and head pose velocity:

$$\mathbf{\Sigma}_{est} = \begin{bmatrix} 0.0199 & -0.0055 \\ -0.0055 & 0.0196 \end{bmatrix}$$

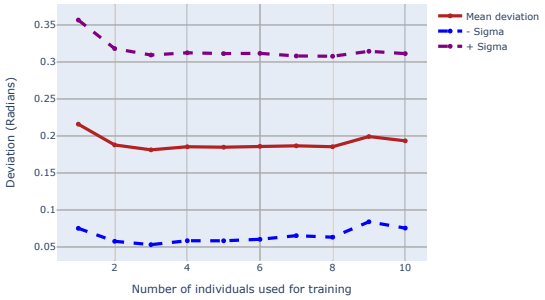
The matrices above only differ slightly, and the standard deviations σ_{xx} and σ_{yy} for both covariances becomes ≈ 0.14, which is expected as it lies within one standard deviation in the tests displayed in Figure 3.6.



(a) id64



(b) id76



(c) id86

Figure 3.6: Accuracy test of the mean error between the estimated gaze and the true gaze point done on three different individuals using only head pose as input. The x-axis denotes how many individuals the model has been trained on and the dashed lines displays one standard deviation from the mean value.

The results of this comparison can be seen in Figure 3.1, where two versions of the regression model were implemented; one using only head pose direction as input, and one using both head pose direction and head pose velocity. The plots displayed are the distributions of probability densities of the true gaze directions $\mathbf{x}_{true}$ in the corresponding covariances generated from the gaze estimators $\boldsymbol{\mu}_{est}, \boldsymbol{\Sigma}_{est}$ as $\mathcal{N}(\mathbf{x}_{true}|\boldsymbol{\mu}_{est}, \boldsymbol{\Sigma}_{est})$. What can be noted is that the performance of both of the regression models is more or less equivalent to the grid at the higher spectrum of Q_{gaze}'s. When the Q_{gaze} gets lower however, the regression model starts to become more accurate.

This largely has to do with the difference in handling the periphery in the implementations. The grid model has a different distribution for each bin, and this distribution tends to become larger in the periphery where the gaze quality worsens. This is despite the use of the gaze multiplier (equation 3.1) when training the grid correlator. The regression models on the other hand create a more continuous relation between the input and output, as well as having a fixed size of the distribution for the entire state space based on the covariance training mentioned above.

Another thing that could be noted was that velocity does not seem to provide better performance. This is most likely due to the simple assumption of just adding the head velocity to the input is a simplified model of the behavior that fails to provide any additional useful information. To be able to utilize this potential, more time needs to be invested in researching behavior patterns and what other assumptions can be made to improve the gaze estimate with the MLP regression model.

With this data in mind, it becomes clear that in the purpose of gaze estimation when the gaze quality is sub-par, the regression model outperforms the correlation grid in the accuracy department. It also has greater potential when it comes to further development, as the head pose velocity could provide more valuable information if it is implemented correctly. This demands more time dedicated to behavior studies on head movement, however. Therefore the regression model using head pose direction as input was chosen as the candidate for gaze estimation.

3.2 Clustering Methods

To compare the performance of different clustering methods, a basis for this testing needed to be defined. Important features to compare include the accuracy of the clustering itself, the speed of the prediction and update steps of the algorithms, as well as the ease of correlating the clusters to the points of interests in the vehicle interior. Although the latter metric is not quantifiable, it is of great importance as some methods do not allow for manual initialization of the clusters or even the number of clusters in some circumstances. This can subsequently lead to a difficult task of correlating the clusters to the physical objects of interest inside the vehicle.

This directly eliminates both the K-means algorithm (see section 2.3.1) and the Fuzzy C-means method (see section 2.3.2) as potential clustering methods, as the robustness based on replicability will be very low and due to the dynamic behavior of the data and the individual differences between drivers, this will lead to an unpredictable algorithm that is hard to obtain the intended information from. This is also a part of the reason for not carrying out any large scale tests on the DBSCAN (see section 2.3.5) and CDBSCAN (see section 2.3.6) algorithms, together with the fact that these methods required large amounts of data to be able to form any clusters. Therefore, these methods were also deemed not suitable for the purposes of this thesis.

The most obvious way to compare the different clustering methods became (be-

cause of the reasons mentioned above) to firstly visually judge the results and to what extent they might resemble the sectors of interest in the vehicle. After this, a quantifiable comparison was done by analyzing the classification of every point compared to what interior object their corresponding 3D gaze vector intersected with, as well as the handling of uncertainties in the model.

The comparisons between the 3D intersection points and the cluster needed a benchmark to determine if the results can be considered to improve the accuracy or not. Therefore it was decided that the raw SEP data would be used with each clustering algorithm to get a baseline confusion matrix describing what type of intersection points got classified as what object and thus which performance can be expected from each method. The chosen algorithm will thereafter be implemented to its full extent together with the chosen gaze estimator and update scheme, after which the same confusion matrix will be generated with the less accurate TC data to compare the performance.

For testing of the clustering algorithms, the input data used for clustering was the gaze direction data from SEP. After the clustering was performed, the estimated clusters inherencies were then compared to the intersected 3D object of the vehicle from the 3D gaze vectors, using SEP for maximum accuracy. Thereafter an analysis if the gaze direction measurement was classified as the correct object or not was carried out and the results were displayed in confusion matrices, which shows the percentages of all the points intersecting with a certain object being classified as each object.

A common and expected issue with the chosen fitness metric is the lack of a "true" ground truth in the matter of what object is being observed at each data point. The solution of taking the most accurate data and using it to find the 3D intersection points with the model of the car thus introduces a few common errors and faulty classifications that have nothing to do with the clustering algorithms themselves. Due to the limited scope and time frame of this thesis, no time could be allocated to create a better ground truth for this purpose, as the only "perfect" solution would have been to manually annotate about 8 hours of video recordings.

With this in mind, a few errors in the confusion matrix should be expected. Firstly, the physical area classified as the EOR is substantially smaller than the actual area corresponding to the driver looking at the road. This leads to a substantial amount of data points that actually are EOR-points, to have their respective ground truth objects being the windshield. Secondly, the cluster corresponding to the front windshield is located on the left side of the window (see Figure 3.7). The purpose of this is that the main part of the gaze points located on the left side are either EOR or located on the inner mirror, as well as the only distinguishable points where a person is looking through the front windshield and not on the road is on the left side.

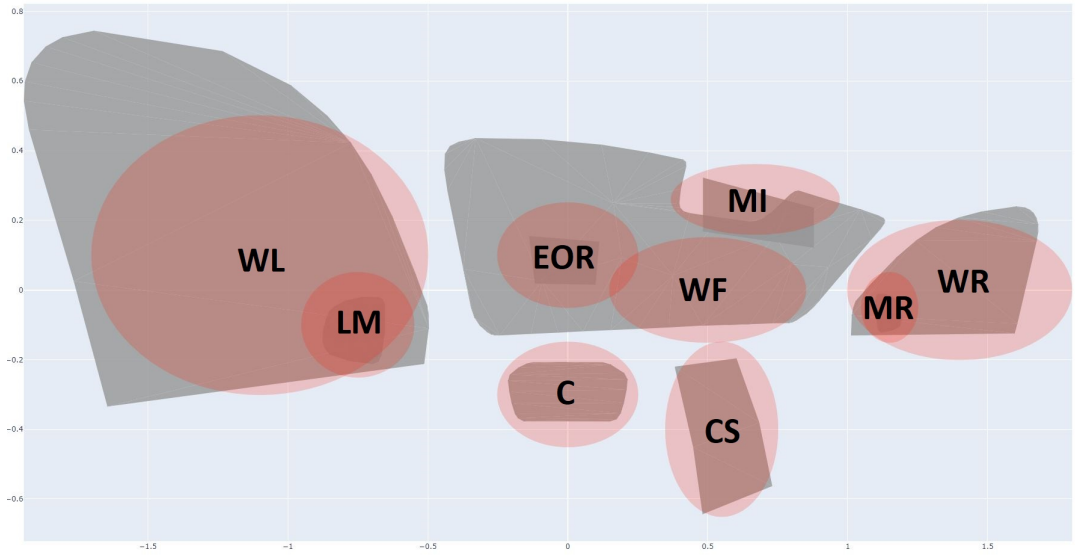


Figure 3.7: Plot of the sizes and locations of the initialized clusters overlayed over a simplified projection of the car interior consisting of the objects of interest.

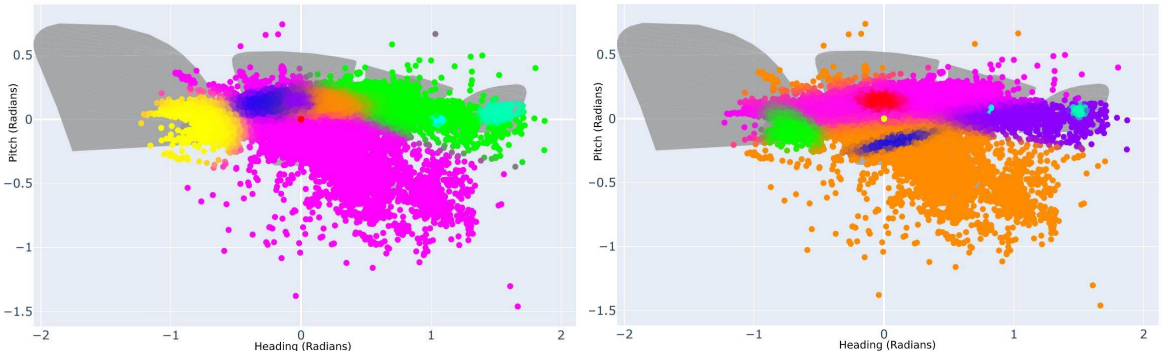
These two things result in a large number of points having their ground truth set as the front windshield but predicted as being EOR when the majority of these points are mislabeled due to the ground truth generation not being perfect.

In addition to this, the clusters corresponding to the side mirrors are larger than the physical mirrors themselves. This is due to the accuracy of the non-calibrated system not being perfect, which leads to many points having their respective ground truths set as the side window but predicted as the side mirror. As mentioned above, this is also a behavior that stems from the ground truth generation, and all of these mentioned behaviors should be regarded when looking and analyzing the results below.

3.2.1 Expectation Maximization

During the process of testing and evaluating the Expectation Maximization algorithm, it was initialized and trained the same way for each case, and evaluation was carried out with the same data. This was done to keep the results as comparable as possible. Additional numerical results than the ones displayed here can be found in Appendix A.2.

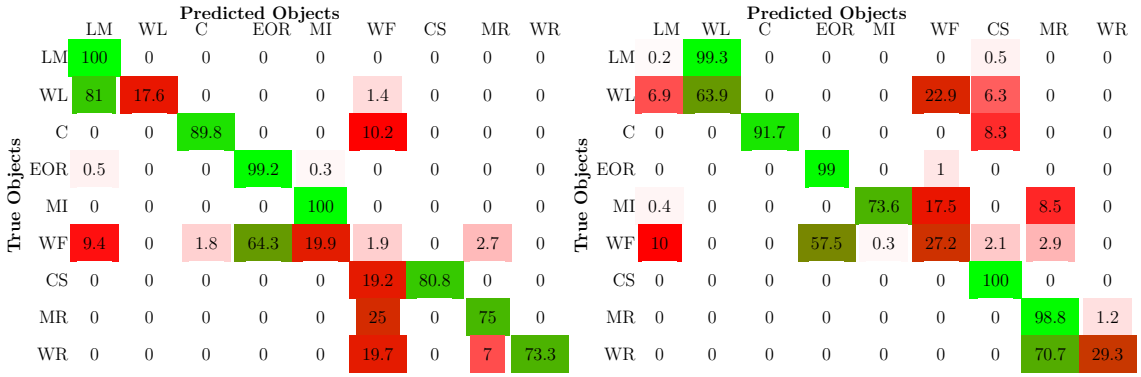
One of the largest contributing factors to the apparent success of the EM algorithm was the ability to initialize the locations of the cluster themselves. This was very important for the purpose of this thesis, as the structural integrity of the locations of the interior points of interest was ruined if the clusters were not placed and sized according to the general locations of the objects to be detected.



(a) Without Mean Initialization **(b)** With Mean Initialization

Figure 3.8: Plot of clustered points with the color corresponding to the cluster inherency generated from the standard EM algorithm

Simply initializing the means and covariances of the clusters did not happen to be quite enough to solve the entire problem, however, as the algorithm still has the ability to morph the covariances to find a local maximum according to equation 2.20. This phenomenon can be seen in Figure 3.8, where simply initializing the means and covariances in the correct locations does not ensure convergence to a model representative of the actual locations of the objects. This is also visible in Figure 3.9, where the results are not terrible, but in areas such as the right rear view mirror and right window, the effect of the morphing of the covariances becomes prevalent. For some data collections, the clusters can also totally converge on the wrong data, which can be seen specifically on subfigure (b) in Figure 3.9 where the vast majority of the left rearview mirror data points were classified as belonging to the left window.



(a) id59 **(b)** id86

Figure 3.9: Confusion matrices of raw SEP data clustered with the EM algorithm

The biggest issue with the EM algorithm was however that it requires a substantial amount of data belonging to each area of interest at every update step. If this is not the case, the cluster connected to an area in the vehicle that is more seldom visited will start to morph and converge on another group of data points, as the algorithm cannot adjust the number of clusters used. This causes divergence in the

model from the actual geometry of the interior, which is difficult to avoid.

This was the main reason for the construction of the GBG algorithm (see section 3.2.2), where the covariances are kept fixed and each point only affects its inherent cluster. This becomes more suitable for live use where it cannot be expected to have access to data where all areas of interest are covered.

In addition to this, the same initial conditions might not result in the same solution twice with all affecting factors being the same, which inhibits the reproducibility and the reliability of the system.

3.2.2 Gaussian Binary Gravity

To combat the prevalent issues in the EM algorithm, the GBG algorithm was introduced, and based on the results of the two methods, this seemed to come out on top. Firstly, the issue of the clusters losing connection with their respective physical objects of interest is not prevalent in GBG (see Figure 3.10 and 3.11). This was largely due to the change of the update step of the clusters, as well as how the covariance is handled. This procedure is explained in section 2.3.4.

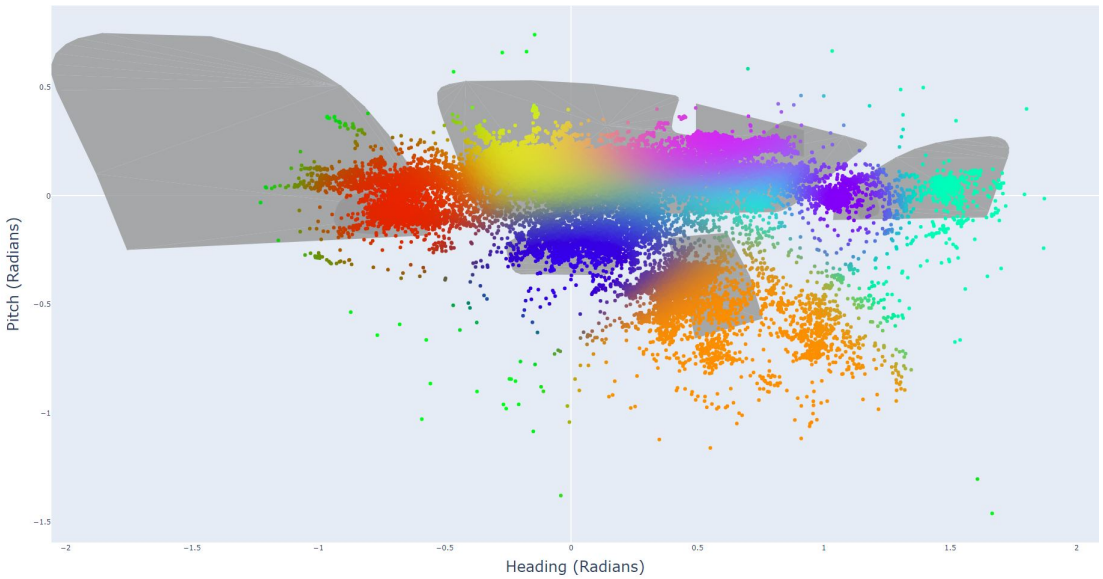


Figure 3.10: Plot of clustered points with the color corresponding to the cluster inherency generated from the GBG algorithm

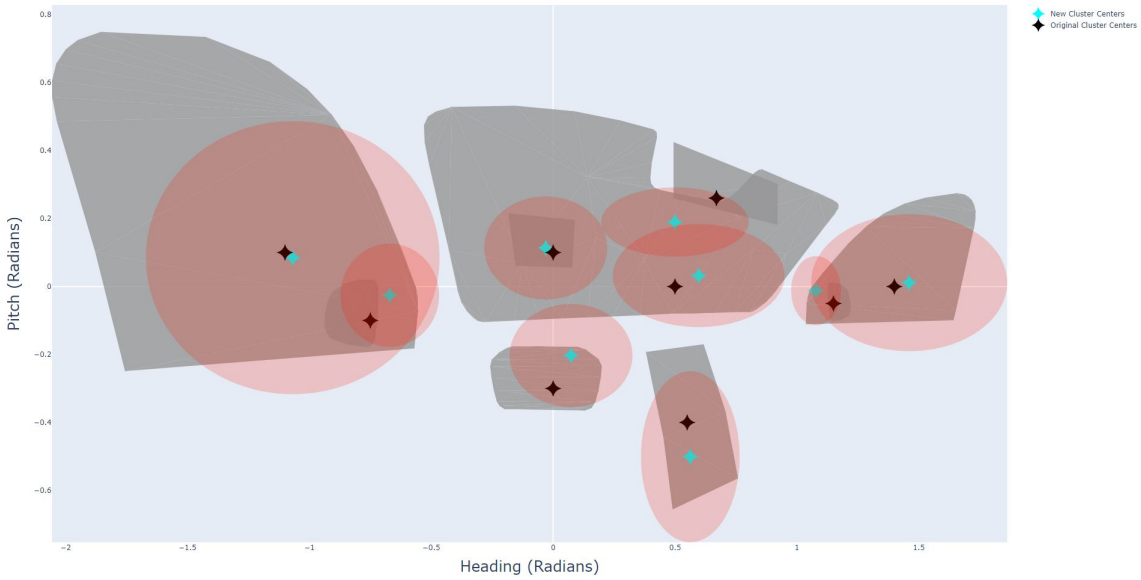
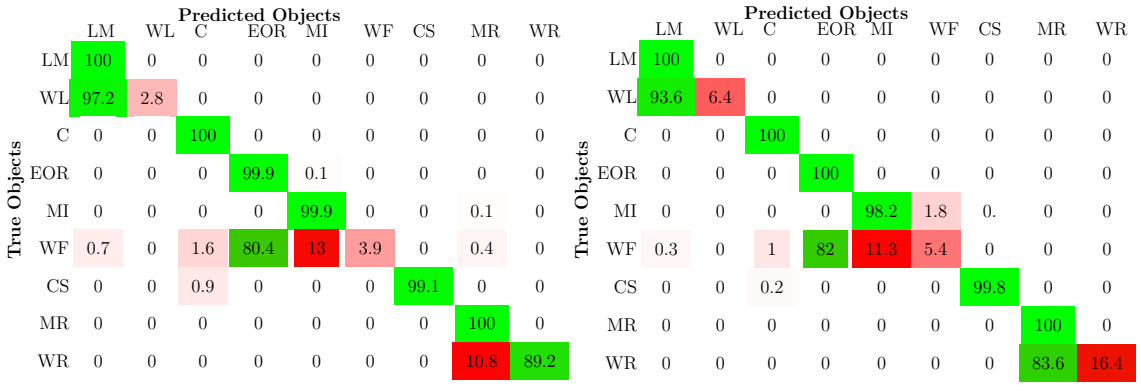


Figure 3.11: Plot of new (turquoise) and old (black) cluster centers generated from the GBG algorithm

Mostly, the results were promising, as can be seen in both subfigures of Figure 3.12 (more results can be found in Appendix A.2). What can be noted is that the issues of the false positives in these matrices are largely due to the questionable ground truth that was discussed at the beginning of this section. Despite that, the areas around the periphery tend to be more prone to misclassification. This is mainly due to the gaze quality plummeting in those areas of the state space (see Figure 2.5), but also a result of the current cluster initializations. With more time allocated to tuning, this is mendable through an iterative process to find an optimal initialization point for each cluster, or perhaps an automatic placement based on the head position of the individual. This was neglected in this project due to the quite limited time frame available.



(a) id59 **(b)** id86

Figure 3.12: Confusion matrices of raw SEP data clustered with the GBG algorithm

With this said, comparing the performance between the different clustering algo-

rithms discussed in this thesis, the GBG method is by far the most suitable for this project. The performance itself is quite good when the data is of good quality, which means that it can distinguish between the necessary areas of interest provided decent data. It also provides robustness in the aspect of maintaining the structural integrity of the vehicle interior, which leads to the usability when data in all areas of interest cannot be expected at all times during real-time use. Due to these reasons, the GBG algorithm was chosen as the candidate for clustering.

3.3 Cluster Update Handling

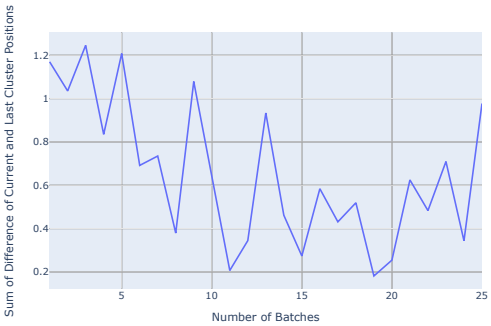
In real-time use, the update of the clustering algorithm introduces a few issues. First of all the updates need to happen continuously in order for the model to gradually adapt and converge for each individual driver. Secondly, to save every single measurement indefinitely to provide the clustering algorithm with the proper information to converge is not feasible, as storage is limited and computational time will increase as more information is collected. This is the reason for the need of a handling scheme for the update of the algorithm.

To determine how to handle this update, a few tests were carried out to gain knowledge of how the chosen algorithm behaves during sequential updates. The results of this have the main purpose of being a guideline of how fast the algorithm converges, and consequently how to handle both the amount of data needed for the model to be maintained as well as the importance of not impacting the computational strain and storage strain on the system in an unnecessary manner.

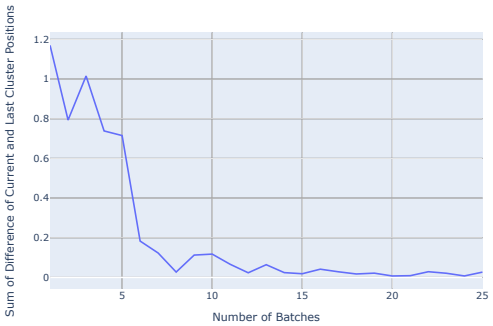
Firstly, to simply perform a clustering update for every single sample is not a solution to the problem of needing to save all collected measurements to perform clustering that converges correctly. In this case, the model itself will fully be affected by one single sample when the algorithm itself is constructed to work in a probabilistic manner which requires a set of data to be effective. This will lead to the model never converging and the accuracy of the clustering will be questionable at best. On the other hand, not updating the model until the current driving session is over will strongly impact the live performance, mainly during the first driving session, as the default values of the cluster locations will be used for prediction during this session. It will also create issues with future constraint implementations, as each update will be done on all data at once, which will make it harder to detect when a constraint is violated and thus how to handle it.

The solution is to instead have batch updates, which simply collects all recorded samples during a smaller period of time starting from the last update and using this set of samples to update the model. During this project, one batch was defined as 3,600 data points, which on average corresponds to 60 seconds of video data. To deduct what amount of data the chosen algorithm needs to converge and maintain this geometrical relationship, batch tests were performed. In Figure 3.13 it is visible that using only one batch does not lead to any convergence at all (see subplot (a)), as the cluster movement fluctuates significantly. It is also noticeable that when

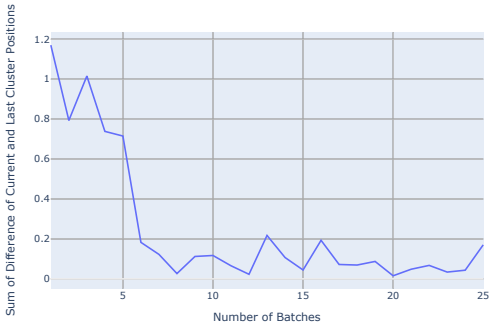
keeping all batches (see subplot (b)), the model tends to converge at around 12 batches. With this information, a test using a batch memory of 12 batches at all times was carried out, which subplot (c) displays. This provides almost the same convergence amount, while not needing to keep all data points for every update step. More figures that reinforce this conclusion can be found in appendix A.1.



(a) Single Batches



(b) Cumulative Batches



(c) Max 12 Batches

Figure 3.13: Plots of the difference between clusters between batches with different handling of batch storage for the data file id59

3.4 Statistical Correction with Hidden Markov Model

To further improve the accuracy of the algorithm predictions, particularly when the gaze quality is low, a hidden Markov model was implemented and trained. The hypothesis is that given a sequence of predicted clusters, where *predicted* refers to the cluster with the highest probability for a given data point, the pattern of behavior from previous data points provides a baseline from which the next observed cluster can be predicted. This estimate will then support the gaze estimate extracted from the regression model, which uses the head direction and head direction velocity.

The flow of the correction algorithm is shown in Figure 3.14. Cluster predictions

from the Smart Eye Pro multi-camera system was used to supply the model with the hidden states, which have a higher accuracy in gaze direction, while data from the less accurate single-camera Tracker Core system was used to obtain observable states.

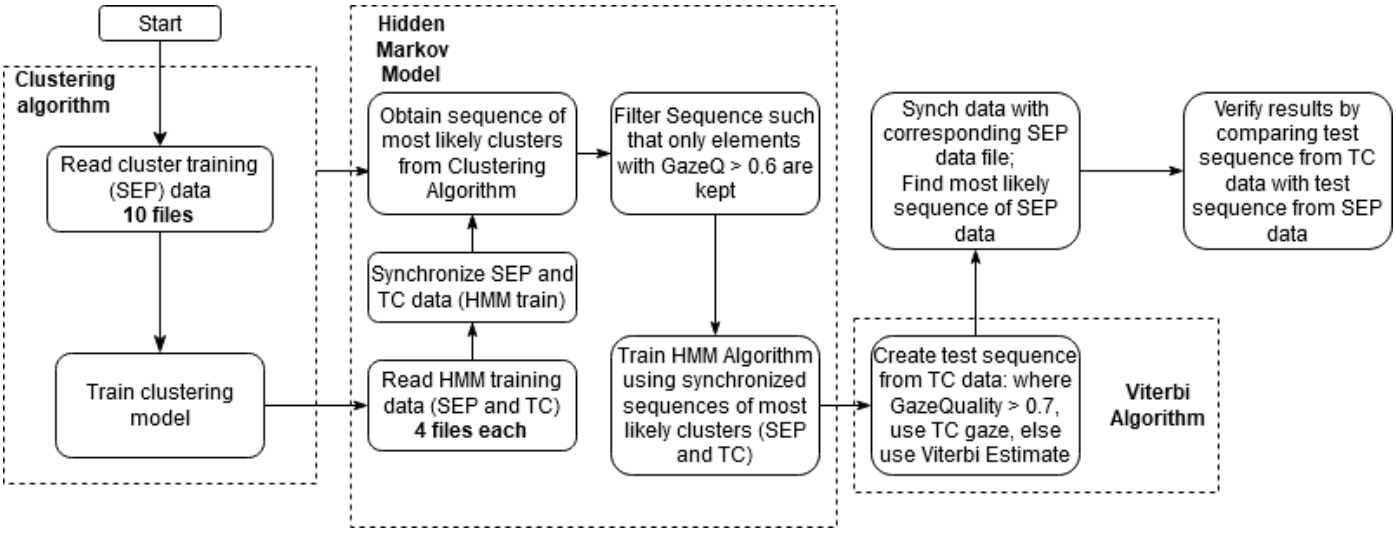


Figure 3.14: Flowchart of the testing process of the hidden Markov model.

The first step was to synchronize the data from the multi-camera system with data from the single-camera system. To verify that the two data sets are synchronized, we look at the time stamp for each frame. Both systems were used simultaneously to record each driving session, however, there were some differences in how these work, which affected the data. More specifically, two problems needed to be solved:

1. Time unit discrepancy. The single-camera system stores time in Unix Epoch time, while the multi-camera system stores time in Windows Epoch time. Therefore, the time unit needs to be taken into account when comparing the two time-columns.
2. Misaligned data. Even after converting the time units, there was a slight mismatch in the data. Therefore, a method was developed which matches each single-camera system data frame to a multi-camera system data frame such that the time difference is minimized.

To address the two issues described above, a method was implemented that converts the time format in the TC files (Unix Epoch time) to the time format in the SEP files (Windows Epoch time). The two camera-systems capture images at the same rate, meaning that the average time difference between any two consecutive frames in the data sets are sufficiently similar that we can assume a linear slope. As a result, it was good enough to simply shift the two lines so that they align. Any items that did not align were discarded. To counter small variations in the frame rate, each item in the SEP data that was matched with the corresponding item in the TC data such that the difference in time was minimized.

The final cluster model was trained on ten multi-camera system-files (SEP), all with different driver IDs. Next, four SEP data files were synchronized with their respective TC data file, such that for every timestamp in each of the SEP data set there is a corresponding data frame from the TC data from the same driver ID. These two data sets were then given as input to the clustering algorithm to find the sequence of most likely clusters. These two sequences were then used to train the hidden Markov model, where the sequence of clustered SEP data was used as hidden states sequence, and the TC data was used as observable states. The three matrices (starting probability, transition probability, emission probability) are shown in Tables 3.1, 3.2, and 3.3 respectively.

Lastly, the test and verification data sets were created by synchronizing a TC data file with a SEP data file and then finding the most likely sequence was for each of these sets found. The process to obtain an improved sequence then goes as follows: For each item in the test TC data sequence, the quality was inspected. If the quality is higher than a threshold, the item was added to the improved sequence. If the data is below the threshold, the improved sequence was given as input to the Viterbi algorithm (see chapter 2.4.3) to obtain a best estimate of the next item, and this was used to replace the low-quality cluster estimate. This was then compared to the verification data set, which comes from the SEP data system.

The performance of the algorithm with the hidden Markov model enabled, trained on different numbers of individuals, was illustrated as a confusion matrix is shown in Table 3.4 along with the baseline case of the unmodified sequence of TC cluster objects in Table 3.5. The results show that the unmodified sequence of TC cluster objects obtained an accuracy of 76.74%, while the sequence which was modified by the Viterbi algorithm after training a hidden Markov model obtained an accuracy of 48.47%.

Unfortunately, this means that the model did not show sufficiently good results to include the hidden Markov model in the final algorithm, as the overall accuracy decreased with this module implemented compared to without.

Possible explanations to the poor results are that this is either due to a lack of high-quality data in the periphery of the car, or due to the fuzzy classification of sectors, or a combination of the two. The hidden Markov model relies on certain data in the hidden sequence and observed sequence to create the matrices later used by the Viterbi algorithm. However, with the GBG clustering algorithm used in this project, the classification of a data point returns a vector of probabilities. It could for example be classified as 50% eyes on road, 20% left mirror, and 30% left window. As the hidden Markov model does not work with fuzzy values, this needs to be approximated as 100% eyes on road. This means that even if the highest probability of a sector from the classification is only 25%, that sector will be treated as if it was 100% in the hidden Markov model. Further, the quality of data also varies, especially in the periphery of the car. Even if the classification of a gaze point returns a single object with 100% confidence, if that gaze point had a low quality we

cannot fully trust that measurement. Any such uncertainty will be reflected both in the clustering, and then again in the creation of the hidden Markov model, this indeed seems to undermine the precision of the output in the Viterbi algorithm. Introducing a quality threshold or a certainty threshold to avoid any low-quality data might increase the accuracy in areas where high-quality data is abundant, but will also lead to a smaller statistical sample which increases uncertainty in regions where high-quality data is scarce.

LM	WL	C	EOR	MI	WF	CS	MR	WR
31.93	0.01	10.79	61.99	9.41	6.68	4.48	3.20	0.25

Table 3.1: Starting probability of the Hidden Markov Model from four synchronized data files.

Object of Origin	Object of Destination: LM	WL	C	EOR	MI	WF	CS	MR	WR
LM	91.78	0.02	3.39	4.74	0.00	0.02	0.06	0.00	0.00
WL	33.33	33.33	16.67	0.00	0.00	0.00	16.67	0.00	0.00
C	0.78	0.01	92.36	3.56	0.04	0.86	2.38	0.00	0.00
EOR	0.25	0.00	0.52	98.21	0.59	0.39	0.02	0.00	0.00
MI	0.00	0.00	0.07	3.85	92.31	3.56	0.03	0.17	0.01
WF	0.01	0.00	1.46	3.67	5.21	87.79	0.84	0.99	0.04
CS	0.03	0.00	5.90	0.29	0.07	1.39	91.83	0.03	0.46
MR	0.00	0.00	0.00	0.00	0.43	2.01	0.12	97.00	0.44
WR	0.00	0.00	0.00	0.00	0.79	1.83	12.57	9.42	75.39

Table 3.2: Transition probability of the Hidden Markov Model from four synchronized data files.

Object in Hidden Sequence	Object in Observed Sequence: LM	WL	C	EOR	MI	WF	CS	MR	WR
LM	88.82	0.02	0.75	10.42	0.00	0.00	0.00	0.00	0.00
WL	33.33	0.00	66.67	0.00	0.00	0.00	0.00	0.00	0.00
C	9.88	0.02	67.05	14.09	0.00	0.68	8.28	0.00	0.00
EOR	1.14	0.00	0.68	97.02	0.02	1.13	0.00	0.00	0.00
MI	0.00	0.00	0.13	15.69	17.45	66.35	0.10	0.28	0.00
WF	0.30	0.00	9.86	12.19	0.16	67.44	7.52	1.02	1.52
CS	0.34	0.00	9.06	0.80	0.00	0.11	89.39	0.20	0.10
MR	0.00	0.00	0.00	0.00	0.00	5.99	18.89	46.67	28.45
WR	0.00	0.00	0.00	0.00	0.00	0.00	31.06	27.33	41.62

Table 3.3: Emission probability of the Hidden Markov Model from four synchronized data files.

Actual Object	Predicted Object: LM	WL	C	EOR	MI	WF	CS	MR	WR
LM	68.81	0.00	7.78	21.16	0.51	0.00	1.74	0.00	0.00
WL	10.81	0.00	13.51	45.95	24.32	0.00	5.41	0.00	0.00
C	9.78	0.00	53.09	23.07	3.84	0.00	10.23	0.00	0.00
EOR	16.55	0.00	18.45	61.74	1.40	0.06	1.79	0.00	0.00
MI	29.20	0.00	21.45	42.59	6.32	0.27	0.17	0.00	0.00
WF	21.93	0.00	32.49	18.07	19.51	1.28	6.71	0.00	0.00
CS	4.38	0.00	24.82	25.33	5.68	0.00	39.79	0.00	0.00
MR	13.60	0.00	12.11	66.67	6.88	0.00	0.75	0.00	0.00
WR	2.72	0.00	18.28	73.20	4.75	0.00	1.05	0.00	0.00

Table 3.4: Confusion matrix showing the predicted objects from the Viterbi algorithm (columns) intersected with the actual objects, as determined by the SEP data (rows). Note that the Viterbi algorithm never predicted any points on WL, MR, or WR.

Actual Object \ Predicted Object	LM	WL	C	EOR	MI	WF	CS	MR	WR
LM	89.65	1.93	1.48	6.56	0.00	0.00	0.39	0.00	0.00
WL	21.62	32.43	0.00	10.81	0.00	05.41	27.03	0.00	2.70
C	10.33	0.55	66.96	13.23	0.23	1.98	6.52	0.04	0.15
EOR	3.07	2.52	2.24	85.43	0.38	1.36	1.89	0.15	2.96
MI	3.60	0.01	22.51	34.06	36.31	0.15	0.15	0.61	0.63
WF	0.57	0.01	12.11	9.28	3.92	64.16	8.03	1.28	0.60
CS	0.48	0.07	2.70	2.65	0.65	5.01	83.97	3.52	0.96
MR	0.00	0.00	0.00	1.05	5.23	7.77	5.38	62.93	17.64
WR	0.00	0.00	0.28	2.44	0.56	1.40	4.75	40.27	50.31

Table 3.5: Confusion matrix showing the predicted objects from the baseline as determined by the TC data (columns) intersected with the actual objects, as determined by the SEP data (rows).

4

Final Algorithm

This chapter presents the final algorithm, the results obtained from it, and compares this to results from other solutions. The algorithm combines the individual methods deemed most successful at their tasks, namely the gaze estimator, the clustering methods, and the cluster update step. The statistical correction step, which uses the Viterbi algorithm on a trained hidden Markov model was not included in the final algorithm due to the decrease in accuracy it caused during testing.

In section 4.1, the algorithm is visualized using a flow chart and discussed in detail. Section 4.2 presents the results from the final algorithm and compares this to other versions of the algorithm.

4.1 Architecture

To create the final algorithm, the structure of the working parts needed to be established. The main workhorse, the clustering algorithm, needs gaze points to cluster, so logically the gaze estimation needed to be incorporated before the clustering algorithm itself. Thereafter the batch update scheme needs to perform the cluster updates correctly. These working parts and how they interact with one another is displayed in figure 4.1.

First, the operation of the final algorithm depending on the gaze quality needs to be addressed. The TC data that the algorithm expects to get as input data is generally less accurate than the SEP data, even at high gaze qualities. Therefore, this uncertainty needed to be accounted for in the system. This was done by modeling each gaze point with an uncertainty covariance, which is then used by the clustering algorithm to draw samples from to obtain the cluster probability vector for the distribution. This distribution is modified depending on the gaze quality by the first building block, the Gaze Estimator, which due to the results presented in chapter 3 was chosen to be the Multi-layer Perceptron Regressor using head direction to estimate the gaze points. A solution for how to implement the regressor was needed however, as the estimate only provides improvement to the measurement when the measurement itself is deemed to be of "bad" quality. Because of this the algorithm needed to take the gaze quality into account when deciding on which distribution to use for every data point, and the flowchart in figure 4.2 shows the decision process for this entire gaze estimator.

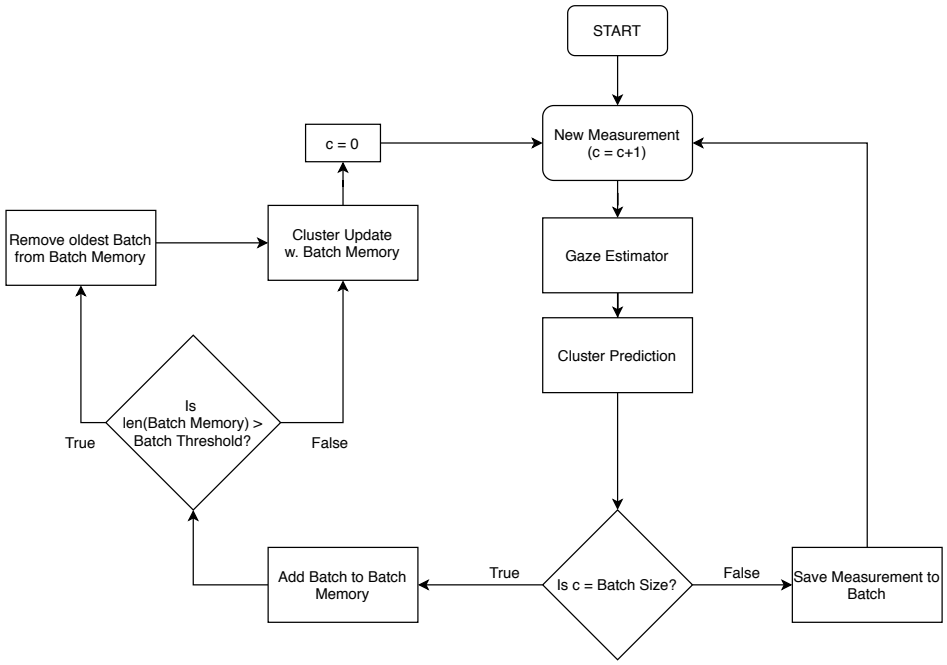


Figure 4.1: Flow chart of the Final Algorithm. c is a counter variable and $BatchThreshold$ is a constant value set before the algorithm is started. Note that this algorithm will run online, and thus has no built-in stopping point.

When the gaze quality is above the set high threshold, the chosen distribution becomes centered in the gaze measurement, while the covariance gets decided by the definition of the gaze cone introduced in Section 2.1.4. When the gaze quality instead is below the set low threshold, the chosen distribution simply becomes centered in the regressor estimate of where the gaze point is expected to be located, and the covariance becomes the estimated covariance that the regressor has calculated according to the explanation in section 2.2.2.

When the gaze quality is between the high and low thresholds however, both the measurement and the estimate need to affect the resulting distribution to improve the measurement with the estimate. This is done by utilizing the so-called covariance intersection algorithm, defined by the two equations

$$\Sigma_r = \Sigma_1(\Sigma_1 + \Sigma_0)^{-1}\Sigma_0 \tag{4.1}$$

$$\mu_r = \Sigma_1(\Sigma_1 + \Sigma_0)^{-1}\mu_0 + \Sigma_0(\Sigma_1 + \Sigma_0)^{-1}\mu_1 \tag{4.2}$$

Where, given two multivariate distributions $\mathcal{N}(\mu_0, \Sigma_0)$ and $\mathcal{N}(\mu_1, \Sigma_1)$ the resultant distribution $\mathcal{N}(\mu_r, \Sigma_r)$ obtains a mean and covariance according to equations 4.1 and 4.2 (see Figure 4.3). Here the gaze cone will be defining Σ_0, the gaze direction measurement μ_0, the estimated gaze direction μ_1, and the estimated gaze distribution Σ_1. This will effectively increase the certainty of the distribution if the gaze point is okay while maintaining a robust behavior and creating qualified guesses when the data is mediocre, as it creates a statistically based combination of the two distributions.

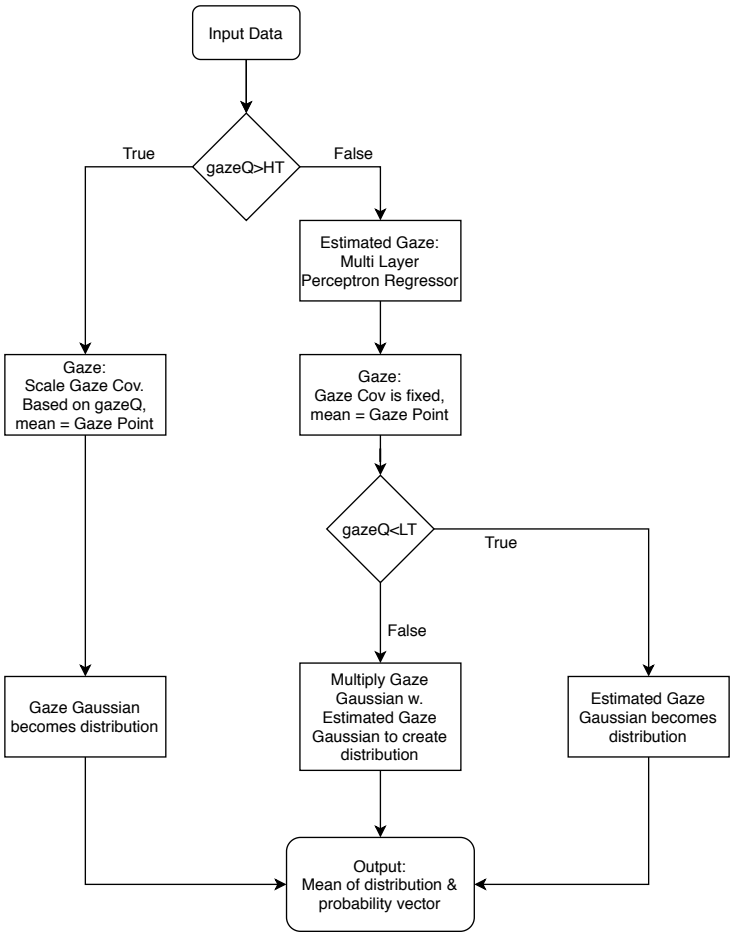
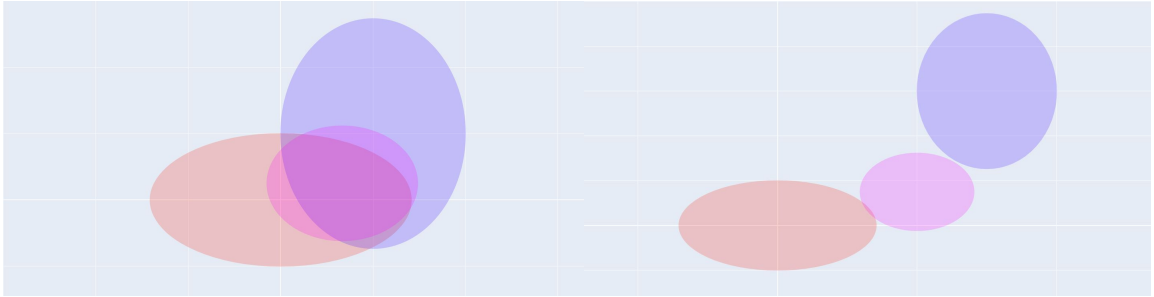


Figure 4.2: Flow chart of how the Gaze Estimator operates within the final algorithm



(a) **(b)**

Figure 4.3: Two examples of the resulting distributions acquired using equations 4.1 and 4.2. The magenta ellipse is the result of a covariance intersection of the blue and red ellipses.

Depending on the gaze quality of the measurement, the corresponding distribution is then used to generate a set number (N) of samples $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ from this distribution. These points are individually tested with the current cluster locations

according to equation 4.3 to obtain the probability vector of each point, where K denotes the number of clusters and μ_k and Σ_k is the mean and covariance of cluster k. These vectors are then summed and normalized in equation 4.4 to create the resulting probability vector for the distribution. This entire process is visualized in a flowchart in figure 4.2.

$$\mathbf{p}_n = \frac{\sum_{k=1}^{K} \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{K} \tag{4.3}$$

$$\mathbf{p}_d = \frac{\sum_{n=1}^{N} \mathbf{p}_n}{N} \tag{4.4}$$

The results are thereafter brought into the clustering algorithm, which has been chosen as the variation of the EM method that has been named GBG (introduced in section 2.3.4), for the prediction step. This is simply the calculation of each measurement's inherent cluster with the current cluster locations. For each measurement, the prediction step uses the Gaussian distribution from the Gaze Estimator, from which n points are sampled and evaluated one by one to each cluster to obtain a normalized probability vector for each point (using equation 4.3). These probability vectors are then summed and normalized to obtain an estimate of the cluster probability for the entire Gaussian distribution with equation 4.4. This procedure can be viewed as a flowchart in Figure 4.4.

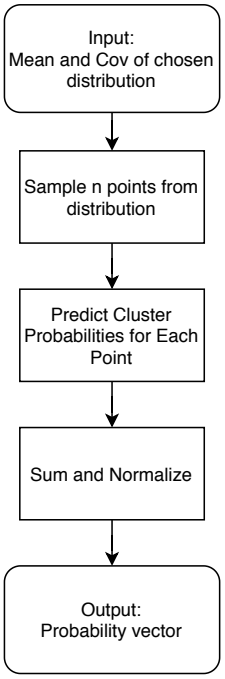


Figure 4.4: Flow chart of how the Clustering Prediction operates within the final algorithm

To handle the update step of the final algorithm, the implementation mentioned in section 3.3 using batches was implemented. To be able to utilize the desired behavior, the storage of batches, and the point where the cluster algorithm update

takes place needed to be decided. Because of this, the final algorithm works in minute batches ($c = 3600$ with a frame rate of 60 Hz) in which each measurement is only predicted, and is then saved into one single batch. When one minute worth of measurements has been collected, the batch is added to what is called the Batch Memory and this memory is used to update the cluster model. This memory contains all the data the clustering algorithm uses for its update step and is used to not let new data single-handedly control the update of the cluster, but instead keep a certain amount of "old" data to maintain the structural integrity of the model. As the results show in figure 3.13, after about 12 minutes the results of the clustering tend to not improve significantly. What can also be noted is that the performance when limiting the Batch Memory to contain 12 minutes of measurements only is affected a small amount compared to the case where all measurements are used for every update.

Due to this, the Batch Memory in the final algorithm has a cap on 12 batches, and at the point when the memory is full and another batch is added, the oldest batch in the memory is removed. This leads to roughly the same performance as saving all measurements without both the computational strain caused by an increased data set as well as the potentially infinite requirements on storage for this algorithm to be able to run.

The final building block is then of course the cluster update itself. To enable the distributions created by the gaze estimator to act as the update points for the algorithm. It was decided to use the mean of the distributions as the point to calculate the difference between the measurement and its inherent cluster as it becomes the most natural reference point. This is because when the gazeQ is above the high threshold the mean of the distribution becomes the actual gaze direction measurement.

The probability vector for each measurement is calculated by sampling from its distribution as described above, and all these factors are used to update the cluster positions according to the GBG-algorithm. A flowchart of the procedure of this final algorithm update step can be seen in Figure 4.5.

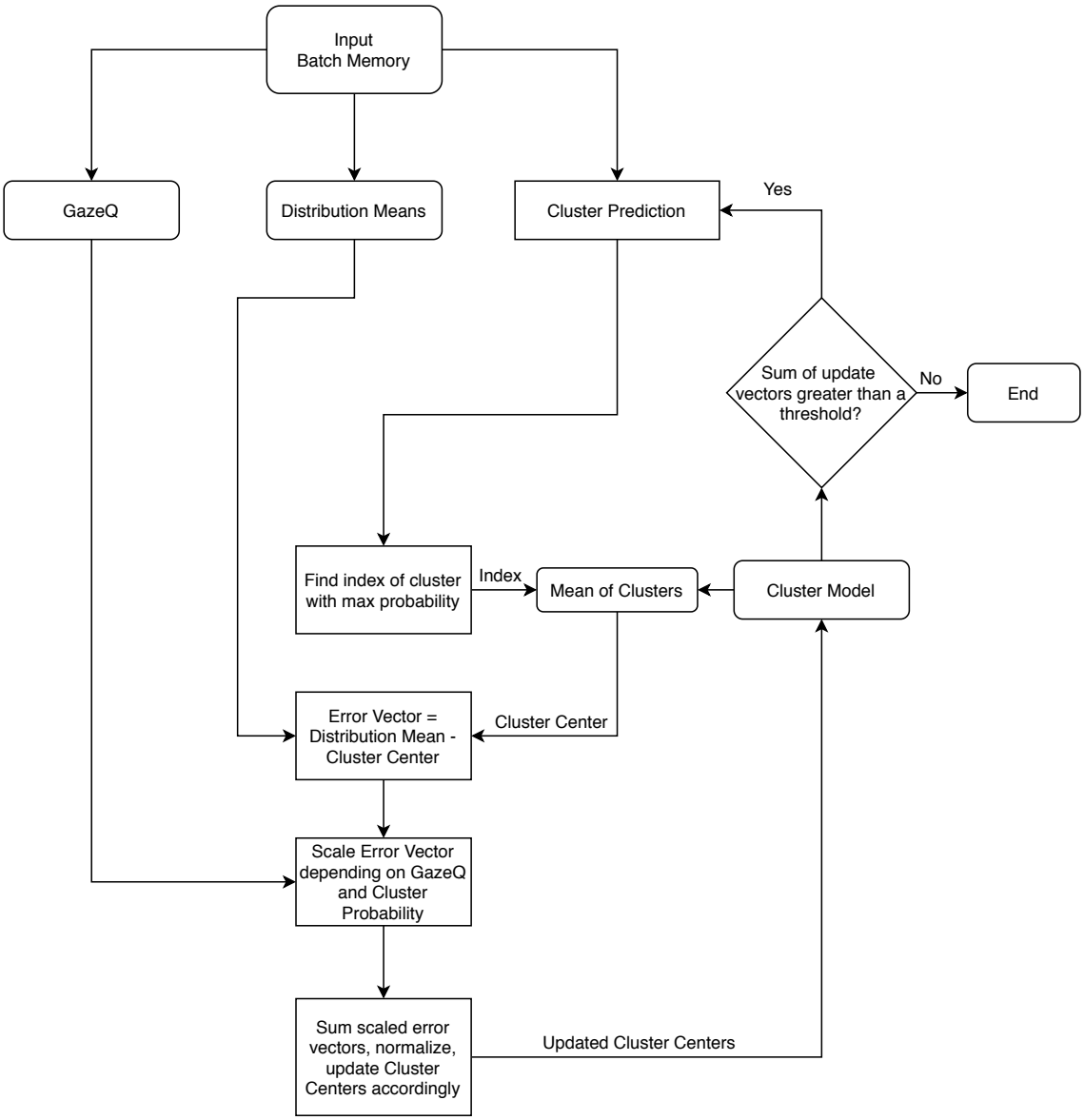


Figure 4.5: Flow chart of how the Clustering Update operates within the final algorithm

4.2 Results

During the gathering of results for the final algorithm, it was decided to use a stepwise integration of the tree working parts in the algorithm to get a clear vision of what performance they provide. The tests used were the same as described in section 3, where the intersections of the 3D vectors from the SEP data on the 3D model of the vehicle interior were used as ground truth. To get a basis of the expected performance on the different test files, the standard GBG algorithm introduced in section 2.3.4 was used to classify the SEP data. The same algorithm was then used on TC data to gain knowledge of how the single algorithm performs on the data that can be expected in real-life use to visualize the differences. Thereafter the gaze

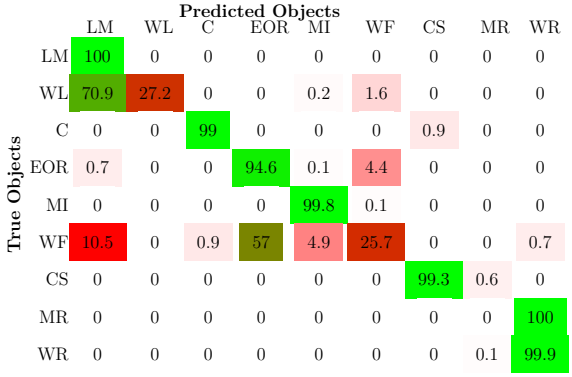
estimator was implemented together with the GBG algorithm and tested with TC data to see how the difference of the introduction of the gaze cone and improving the lower quality points affect the results, and finally, this algorithm was tested using the batch update scheme described in this section. The resulting confusion matrices for the data-id files *id9* and *id51* can be viewed in figures 4.6 and 4.8 respectively, with additional results on other data-id files being found in appendix A.2. Please refer to table 4.1 for an explanation of the acronyms used in the column and row names.

Acronym	**Area of Interest**
LM	Left Rear View Mirror
WL	Left Window
C	Tachometer Cluster
EOR	Eyes On Road
MI	Inner Rear View Mirror
WF	Front Windshield
CS	Center Entertainment Stack
MR	Right Rear View Mirror
WR	Right Window

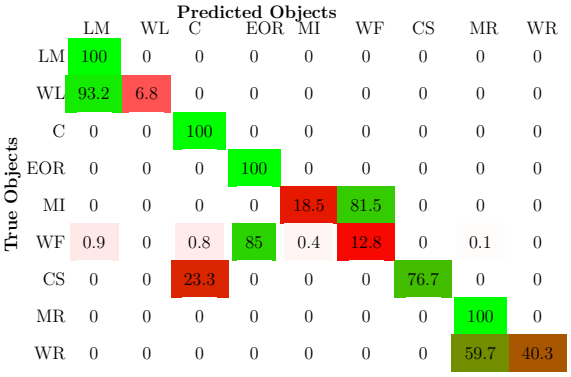
Table 4.1: Table of acronyms used in the confusion matrices

The main focus of these tests is to see how the performance of the different implementations of the algorithm using TC data compares with the standard GBG algorithm using the much more accurate SEP data. This GBG algorithm using SEP data thus becomes the baseline of which performance is desirable, and the results of the other implementations on TC data will show how well they hold up when having less accurate data as well as what issues each method has.

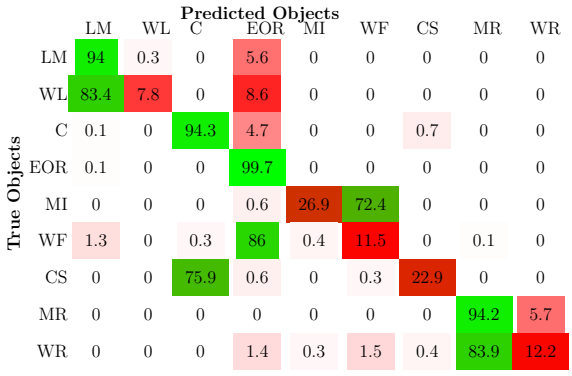
What is immediately notable from these matrices is that the standard GBG algorithm on the TC data performs quite well compared to the one using SEP data. Many of the faulty predictions can be directly connected to the reasons described in section 3.2 due to the ground truth not being perfect. However, in some of the tests, it is prevalent that the inner center rearview mirror tends to be very difficult to maintain a good track on based on the available ground truth when it comes to the TC data. This is most likely due to the TC data's tendency to be more compressed around the x-axis, as can be seen in Figure 4.7, which leads to the difference between the inner mirror cluster and the EOR and front window clusters being much smaller than in the SEP data. It is also noticeable that due to the single-camera setup used to generate the TC data, the periphery clusters have a higher tendency to be miss-classified, especially the right window as it requires a substantial amount of head turn to gaze in that general area.



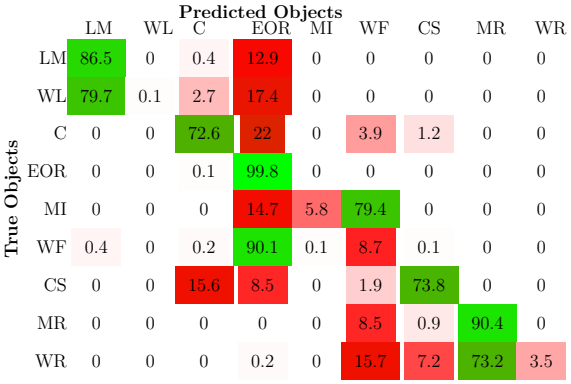
(a) EM algorithm on SEP data



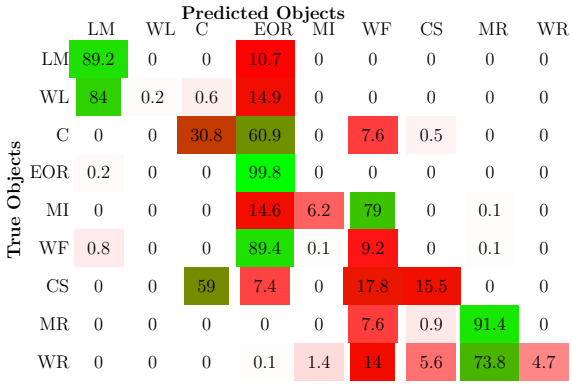
(b) GBG algorithm on SEP data



(c) GBG algorithm on TC data

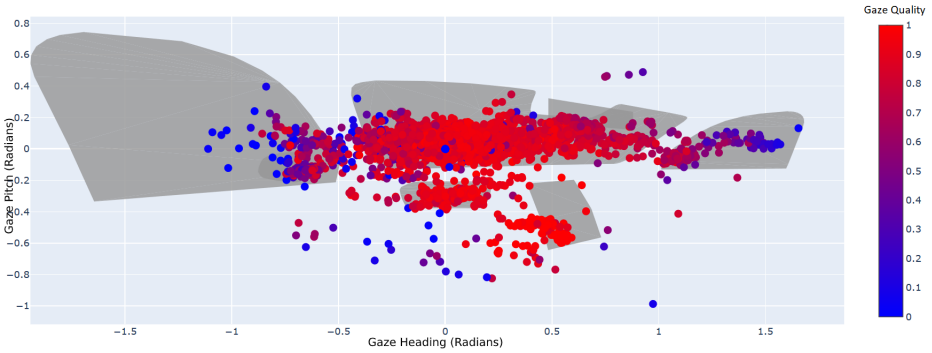


(d) Final algorithm on TC data

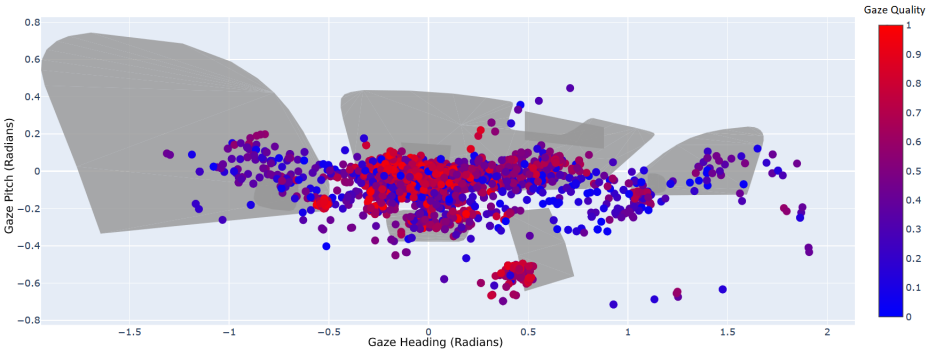


(e) Final algorithm on TC data using batches

Figure 4.6: Confusion matrices generated from the log file id9



(a) 15,000 samples of SEP data with corresponding quality



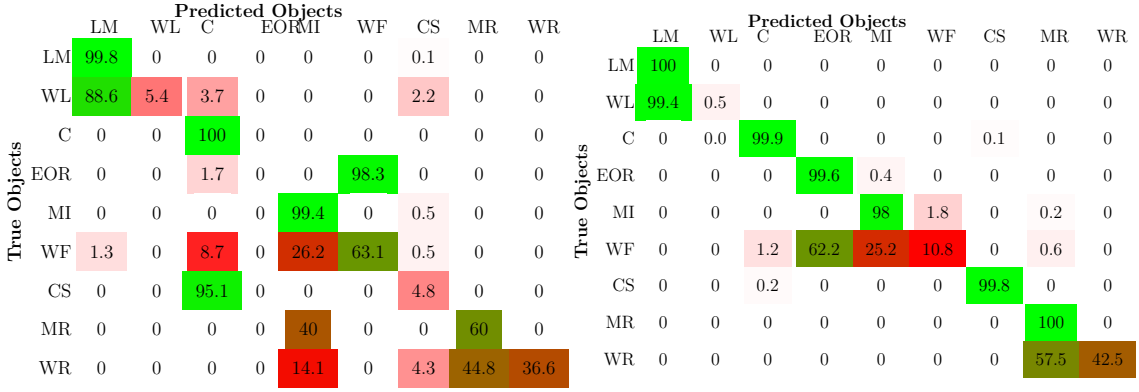
(b) 15,000 samples of TC data with corresponding quality

Figure 4.7: Gaze Heading (x-axis) and Gaze Pitch (y-axis) plotted on top of a 2D spherical projection of the car model around the mean head position of the driver. The data is captured from the same driving session. Note that the SEP data on average is of higher quality, and also that the TC data rarely is of high quality in the periphery.

When looking at the addition of the gaze estimator to the algorithm, some things stand out. The already well-predicted areas tend to be kept well defined, but when the estimator provides additional information in the areas of lower gaze quality (periphery) the estimates are not quite accurate enough to correctly affect the prediction into the true cluster, and therefore slightly worsens these accuracies. This is visible in subfigure (c) and (d) in figure 4.6, where the right window sees a $\approx 4\%$ drop and the right rear view mirror sees a $\approx 9\%$ drop in accuracy. In the cases where the raw GBG clustering incorrectly classifies the areas of interest with negative pitch angles (tachometer cluster and center entertainment stack) however, the gaze estimator provides well defined additional information to correctly separate these clusters. This is most prevalent in figure 4.6, where only applying the GBG algorithm provides 4.7% correctly classified tachometer cluster measurements and 22.9% correctly classified center entertainment stack measurements (see subfigure (c)). When adding the gaze estimator, which is displayed in subfigure (d), these percentages increase to 22% for the tachometer cluster and 73.8% for the entertainment stack respectively. This is a substantial increase in accuracy by being able to adjust the clustering based on the head pose direction of the driver.

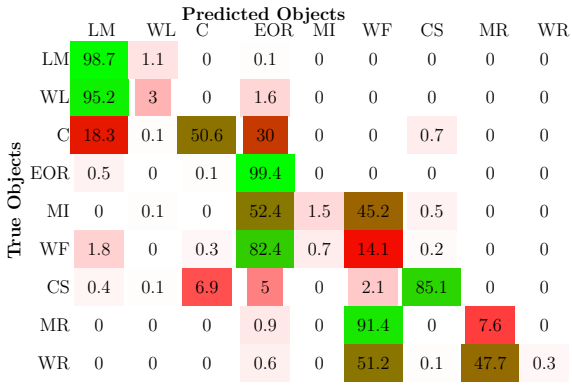
The reason for this behavior is that the training of the gaze estimator only uses data above a certain gaze quality to be able to provide a good and accurate estimate. These points tend to not include the periphery, but extreme pitch angles (y-axis on plots) are not as affected. The TC-data is more prone to bad gaze quality even at larger pitch angles, which is why in the cases where the gaze quality of the data is bad in such areas as the tachometer cluster and center entertainment stack, the performance tends to be improved. This also explains why the model cannot handle the periphery estimates that well. The side mirrors and windows whose correctly classified percentages tend to decrease after adding the gaze estimator, has this behavior due to the estimator not being provided with good training data in those areas. To remedy this, the gaze estimator would have to either conduct sessions where good quality data in these areas are collected to have better overall training data or incorporate a more advanced model of the head pose direction and velocity to utilize more behavior factors to generate a better estimate.

With the batch update added to the algorithm (see subplot (e) in figure 4.8), the model integrity of the GBG algorithm showed its upside. Despite having up to 12 minutes of measurements at one update step to conform the clustering, barley no drop in performance was usually visible compared to the final algorithm without batch updates (see subplot (d)). The one exception to this was in figure 4.6, where the accuracy of for example the center entertainment stack drops from 73.8% to 15.5% (see subplots (d) and (e)), but this is an anomaly as it did not occur in any other test. This means that the structural integrity of the model is kept fairly well with a minimal amount of data needed to be saved, which is highly desirable when real-life use is considered.

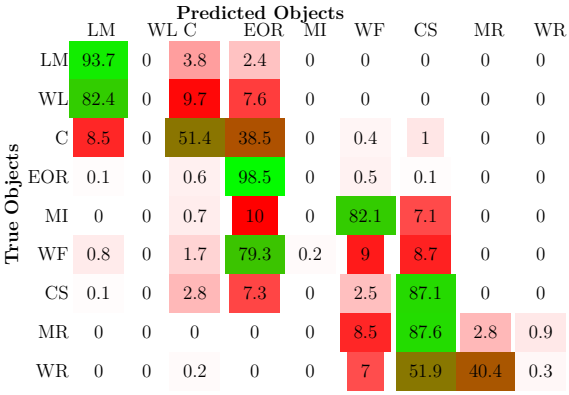


(a) EM algorithm on SEP data

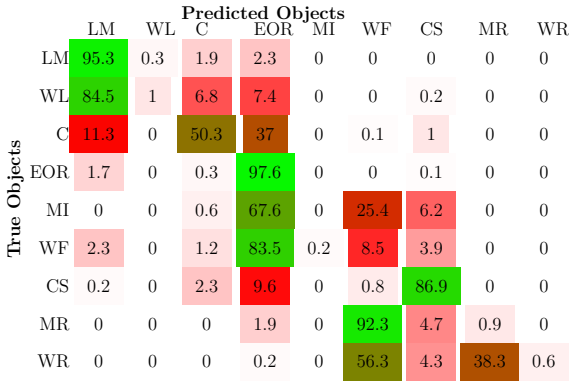
(b) GBG algorithm on SEP data



(c) GBG algorithm on TC data



(d) Final algorithm on TC data



(e) Final algorithm on TC data using batches

Figure 4.8: Confusion matrices generated from the log file id51

To accompany these results two tables (4.2 and 4.3) containing the total clustering times and the total accuracy of the different implementations of the algorithm was also generated. What needs to be noted is that the Final Algo. Batches column in figure 4.2 shows the clustering time per batch as opposed to for the entire process of the data file.

	GBG - SEP	GBG	Final Algo.	Final Algo. Batches
id5	72.6 s	76.7 s	1118.8 s	94.2 s
id9	53.8 s	53.3 s	525.1 s	115.2 s
id12	49 s	50.3 s	749.2 s	88.9 s
id29	41.7 s	41.2 s	2184 s	84.6 s
id51	51.8 s	51.3 s	2400.9 s	144.3 s
id64	46.2 s	43.6 s	984 s	83.3 s

Table 4.2: Table showing the clustering times for the algorithm implementations on the different data files. All algorithms are run on the entire data set except the one denoted with "Batches". All results are in seconds.

	GBG - SEP	GBG	Final Algo.	Final Algo. Batches
id5	40%	39%	33%	27%
id9	39%	38%	36%	35%
id12	34%	33%	31%	32%
id29	39%	39%	36%	36%
id51	28%	27%	24%	23%
id64	35%	35%	34%	34%

Table 4.3: Table showing the total clustering accuracy for the algorithm's implementations on the different data files. All algorithms are run on the entire data set except the one denoted with "Batches". All results are in percent.

What is worth noting in these tables if firstly the immense difference in calculation times between the raw GBG algorithm and the final algorithm. This is largely due to the addition of the need for sampling and prediction from a distribution as apposed to just predicting one point per measurement, which with the sample amount $N = 100$ should roughly lead to a 100 times slower algorithm. In testing the average was fluctuating around 25 times the calculation time which was slightly better than expected. When the batch update scheme was applied, every batch had quite a hefty calculation time despite the max number of measurements having a threshold. This

update time decreases as the model converges however, which leads to the update time decreasing as the model conforms to the driver.

Table 4.3 displays the total accuracy of the different implementations which is quite underwhelming at first glance. These raw numbers become somewhat misleading, however, as for example the front windshield tends to get wrongly classified due to the ground truth definition, and as that area contains a large percentage of the classified points it skews the total statistics a fair amount. But as mentioned at the beginning of this section, this is not the important ground on which the comparison should be done. The comparison should rather be done on the performance of the different implementations using TC data compared to the GBG algorithm using SEP data, as this is the best result that could be generated even with nearly perfect data.

When it comes to using the GBG algorithm directly with TC data, the overall performance is surprisingly similar to the GBG algorithm using SEP data. As discussed above, the performance in the periphery is not nearly as good, but due to the majority of the points the system faces being quite close to origo in the coordinate system, this does not show up on the overall percentages. This leads to the conclusion that despite the drop in accuracy, the GBG clustering algorithm manages to create and converge on a proper cluster geometry well suited for the gaze sector detection faced, given that it is properly initialized. This is a good indication that the process of only affecting the inherent cluster for each measurement prevents an unreasonable divergence when the data quality declines. Therefore, a nearly equal total accuracy can be achieved on the lower quality TC data as on the higher quality SEP data. A drop in periphery accuracy should be expected though.

The implementation of the gaze estimator did not become quite as successful, however. In general, the estimator did provide some valuable information in areas such as the tachometer cluster and the center entertainment stack, but oftentimes decreased the accuracy in the clusters located in the periphery. As mentioned above, more work needs to be applied to this method to obtain reliable information from the MLP regressor. However, the fact that estimates in areas where the training data has been of good quality have provided improving information for a single camera set up, this method shows some promise.

When it comes to the update scheme, the results show that due to the batch size being chosen based on the tests discussed in section 3.3 no substantial drop in accuracy was noted. It shows that it is possible to obtain a converging model that maintains its geometry despite not having access to all data at every time instant, which further shows that the GBG algorithm is well suited for the purpose described in this thesis. As the development of the update scheme has not been the main purpose of this thesis, however, more time could be invested in finding a more suitable approach that is better optimized for the specific purpose of live usage.

As a whole, this proposed algorithm has provided some good input on that the GBG clustering algorithm is well suited for the gaze sector application, even as the

quality is sub-par. It has also shown the difficulties in estimating the gaze based on only a simple head pose direction model, but has provided some indications on that when properly trained an MLP regressor could be used in this type of setting. These can be considered the main takeaways from the tests done in this thesis.

5
Conclusion

This final chapter summarizes the results from the thesis in terms of accuracy (section 5.1) and performance (section 5.2). Also, recommendations for further development are given in section 5.3 along with comments on how results better can be verified.

5.1 Summary

This master's thesis has aimed to solve the task of providing a sector analysis of the gaze data from a driver in a personal vehicle, in order to obtain information on which general area of interest in the vehicle the driver is looking at. Specifically, the task was narrowed down to only handle naturalistic highway driving from a single-camera system when no explicit calibration of the gaze measurements can be expected. This has been done by firstly analyzing the typical behavior of a driver in these circumstances. This provided data on what assumptions could be made in order to simplify the data needed to describe where the driver is looking, as well as what data could be utilized to estimate the gaze direction to compensate for data where the measurement quality is low.

The knowledge gained from this was then used as a basis to test potential algorithms that could perform this sector analysis to be able to group the data into the desired areas of interest. After researching and testing, it became clear that the machine learning branch of unsupervised learning, also known as clustering, was most suitable for this. After some further consideration, a new algorithm based on the Expectation-Maximization algorithm was proposed as the best candidate for this purpose.

To follow this, a few models to improve the performance of this clustering method when the quality of the measurements are not perfect were tested and implemented. The main candidates performed quite well in isolated tests, but the most efficient one that provided good information became the Multi-layer Perceptron Regressor model, which with information of the head direction can predict where the corresponding gaze direction probably lies. The gaze quality generally decreases as the gaze direction diverges from the center, which can be explained by the fact that the driver then is facing away from the single camera. Hence it was necessary to find a method of approximating the gaze direction for data frames where the gaze quality is low.

These separate algorithms were then merged, with the addition of an update controller using batches, to obtain a final proposal of an algorithm. This algorithm provides predictions on what area of the vehicle the driver is currently looking at, with a probability for each of the sectors at every time frame.

5.2 Performance

As the performance of the proposed algorithms in this thesis is concerned, there are both positive and negative takeaways. The constructed clustering algorithm Gaussian Binary Gravity (see section 2.3.4) provides a robust way of retaining a predefined geometry in the clustering and opens the door for introducing more advanced constraints into the realm of clustering than what currently exists. The raw performance of this implementation shows quite good results on the lower quality single-camera TC data when compared to the multi-camera SEP data, and provides a stepping stone for a real-life implementation in a consumer platform. The average total accuracy drop when using this clustering method on TC data compared to SEP data is only $\approx 0.66\%$, which is promising for the method itself. However, the local accuracy in the right periphery areas tends to drop with between approximately 5% and 90% depending on what quality and coverage this periphery area has in the TC data used. The average lies at an accuracy drop of $\approx 30\%$ for the right periphery when looking at the results generated in this thesis, which is not desirable for a real-time implemented system.

The implementation of the gaze estimator in the final algorithm aimed to remedy this behavior but had mixed success in doing so. As explained in section 4.2, the areas improved by the gaze estimator (tachometer cluster and center entertainment stack) were improved because the quality of the SEP training data for the estimator was good in those areas. On the other hand, the areas which saw the accuracy decline as a result of the gaze estimator implementation (right and left mirrors and windows) were not covered by high accuracy data in the SEP training data, which led to the model not being properly prepared to provide high enough estimates in those areas. As this was the most affected area of the accuracy drop in the clustering, this method does not provide sufficient performance with the implementation done in this thesis. The potential is still present for the MLP regressor however, as improving the training data and studying more advanced ways of using data for gaze estimation will most likely improve the performance of this method.

The findings regarding the driver tendencies for gaze directions based on the head movement is also a usable piece of information that is not only important for this particular purpose but also for the whole field of driver monitoring and could potentially provide more knowledge on the behavior of the driver in some situations. The fact that the driver tends to always lead their general attention with the gaze, and that the head pose then follows could be a useful feature when constructing more advanced models on driver behavior in the future.

The implemented batch update scheme was simple but effective and the results illustrate that convergence can be achieved within a reasonable computational cost. When it comes to data amount, this method displayed that it is possible to obtain a converged model with the equivalent of only ≈ 12 minutes of data saved at any given time instance. This is paramount for future live implementations, where it cannot be assumed that exceedingly large amounts of data can be stored. This also affects the computational times, which without the batch threshold will keep growing for each update instance. However, even with the batch threshold of 12 batches with a batch size of 1 minute (3600 samples at 60 Hz), the computational time for each update is longer than the time between update steps (> 60 seconds, see table 4.2). This means that the algorithm will lag, which would not allow it to work in an online setting. This is largely because no speed optimization of the code has been done, as it was not the purpose of this thesis to have the algorithm being ready to run live.

With this said, the results of the final algorithm highlight the issues of trying to model complex human behavior into a fast and usable algorithm. The far from perfect periphery performance in testing is a result of the simplifications made in this thesis, which leads the authors to believe that in order to gain a more accurate gaze sector analysis the focal point needs to be to study driver behavior and create a more advanced model to be able to estimate the gaze in sub-optimal conditions and areas when a single camera setup is used.

The algorithm as a whole is not in an online, real-life usable state due to the low accuracy in the periphery and the update times being too long, but an important step has been taken in the direction to be able to extract valuable data from the driver using a single camera setup. With more time invested in an update scheme that is more time-efficient, an online operating version could be implemented and tested further, which could significantly aid further development of the method.

5.3 Further Development and Implementation

Although this thesis has taken a first step towards a usable implementation of a gaze sector detector without the need for calibration, it is not yet complete and ready for commercial use. However, the method is general enough - it only needs to be initialized once for each car model, and this should be done before the end-consumer is involved - and thus it is easy to customize for new models.

5.3.1 Additional Development

As the results of this thesis have been gathered, combined with the untouched areas of development that did not fit into the time frame of this project, a few interesting and important points on which further development should be carried out have arisen. Most obvious is the already mentioned improvement of the gaze estimator, as this is duly needed to create more accurate predictions when the gaze quality is low.

Other areas that have potential is for example the addition of geometrical constraints to the clustering algorithm itself, as creating a relationship between the different clusters to further maintain the relative relationships based on the geometry of the vehicle interior.

5.3.2 Testing & Verification

A big flaw during the work on this thesis is the lack of reliable verification data available to the authors. Many results tend to become hard to conclude from as the ground truth generated is somewhat flawed as it is not perfect. This leads to many performance results having to be excessively explained before a qualified conclusion can be drawn.

Training data is also a big point on which improvements could be made. For the gaze estimator to provide the system with good quality estimates in areas with low gaze quality, training data with high gaze quality in these problem-areas needs to be collected. Using a multi-camera setup like SEP, this could be done through controlled tests and thus give the estimator a better chance of translating head information to gaze direction data.

When it comes to testing, the addition of a real-time testing rig to perform live testing on the algorithm would have provided a lot of important and useful insights. Due to COVID-19 being prevalent during the execution of this thesis, together with the limited time frame, made this somewhat impossible. Many of the reoccurring and hidden issues could, using such a rig, be more easily located and mended. This would also prepare the algorithm for live use more quickly.

Bibliography

[1] Society of Automotive Engineers, "J3016: Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems." Accessed: 2020-05-29.

[2] European Council, "Pe-82-2019-init."

[3] European Parliament, "European parliament information sheet, grow.a.1.dir."

[4] European Automobile Manufacturers Association, "Economic market report - eu automotive industry 2019."

[5] Smart Eye AB, https://smarteye.se/automotive-solutions/, "Smart eye driver monitoring systems,."

[6] United Nations General Assembly, "Sustainable development goals," 2015. `https://sustainabledevelopment.un.org/`.

[7] H. E. Driver and A. L. Kroeber, *Quantitative expression of cultural relationships*, vol. 31. University of California Press, 1932.

[8] M. Chuang, R. Bala, E. A. Bernal, P. Paul, and A. Burry, "Estimating gaze direction of vehicle drivers using a smartphone camera," in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 165–170, June 2014.

[9] Y. Horiguchi, T. Suzuki, T. Sawaragi, H. Nakanishi, and T. Takimoto, "Extracting train driver's eye-gaze patterns using graph clustering," *IFAC-PapersOnLine*, vol. 49, no. 19, pp. 621 – 626, 2016. 13th IFAC Symposium on Analysis, Design, and Evaluation ofHuman-Machine Systems HMS 2016.

[10] S. J. Lee, J. Jo, H. G. Jung, K. R. Park, and J. Kim, "Real-time gaze estimator based on driver's head orientation for forward collision warning system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 1, pp. 254–267, 2011.

[11] A. Srivastava, S. H. Joshi, W. Mio, and X. Liu, "Statistical shape analysis: Clustering, learning, and testing," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 4, pp. 590–602, 2005.

[12] H. Ng, S. Ong, K. Foong, P. Goh, and W. Nowinski, "Medical image segmentation using k-means clustering and improved watershed algorithm," in *2006 IEEE southwest symposium on image analysis and interpretation*, pp. 61–65, IEEE, 2006.

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

[14] M. Gamer, H. Hecht, N. Seipp, and W. Hiller, "Who is looking at me? the cone of gaze widens in social phobia," *Cognition and Emotion*, vol. 25, no. 4, pp. 756–764, 2011.

[15] F. F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65 6, pp. 386–408, 1958.

[16] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, classifiaction," 1992.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] A. Engel, C. Van den Broeck, and C. Broeck, *Statistical Mechanics of Learning*. Statistical Mechanics of Learning, Cambridge University Press, 2001.

[19] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[20] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, no. 3, pp. 32–57, 1973.

[21] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[22] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal Processing Magazine*, vol. 13, pp. 47–60, Nov 1996.

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[24] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

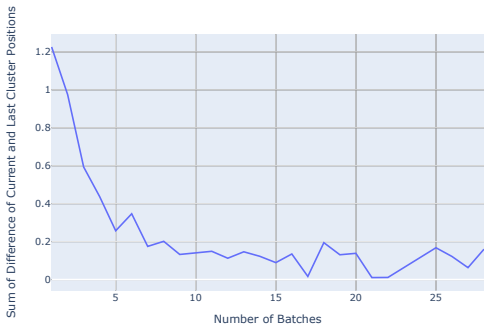
[25] C. Ruiz, M. Spiliopoulou, and E. Menasalvas, "C-dbscan: Density-based clustering with constraints," in *International workshop on rough sets, fuzzy sets, data mining, and granular-soft computing*, pp. 216–223, Springer, 2007.

[26] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

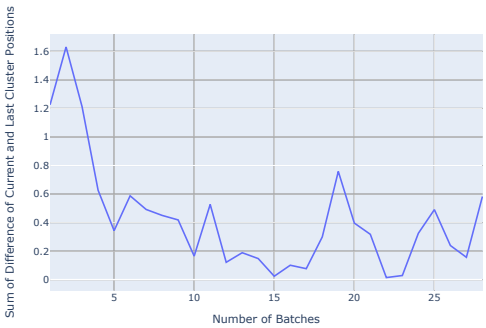
A

Appendix

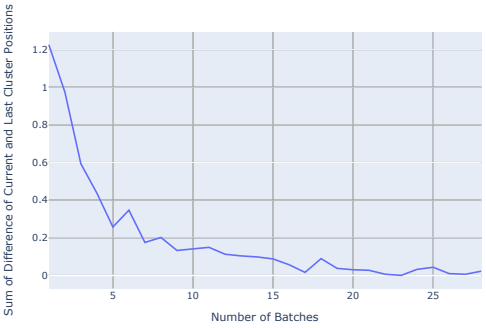
A.1 Batch Handling Tests



(a) Cumulative Batches

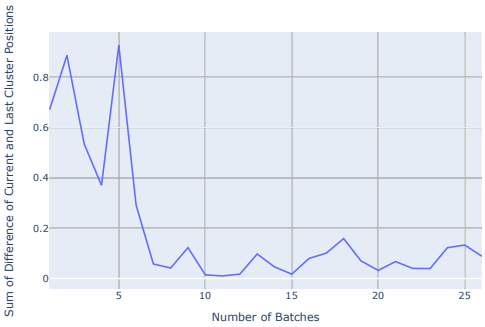


(b) Single Batches

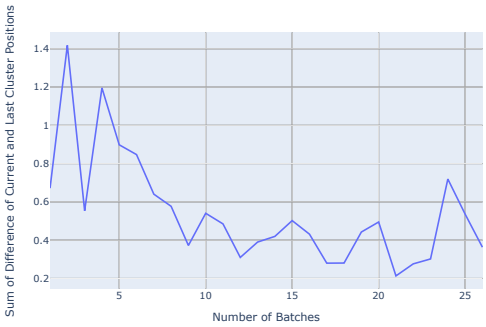


(c) Max 12 Batches

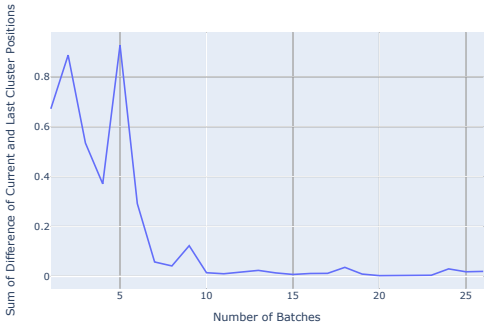
Figure A.1: Plots of the difference between clusters between batches with different handling of batch storage for the data file id9



(a) Cumulative Batches

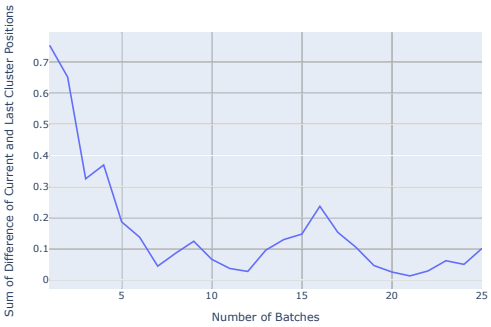


(b) Single Batches

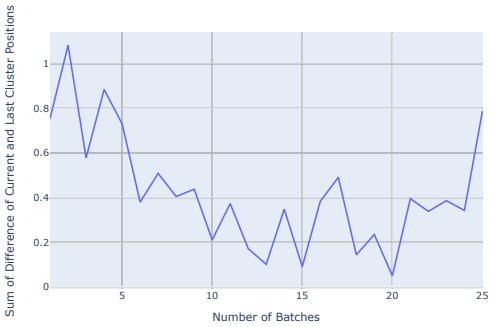


(c) Max 12 Batches

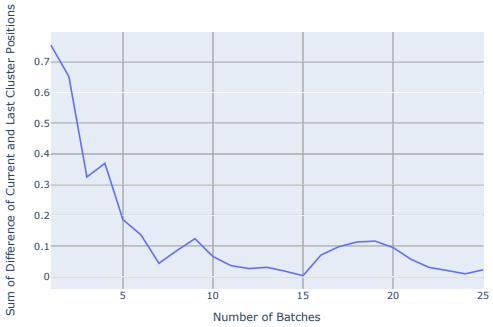
Figure A.2: Plots of the difference between clusters between batches with different handling of batch storage for the data file id12



(a) Cumulative Batches



(b) Single Batches

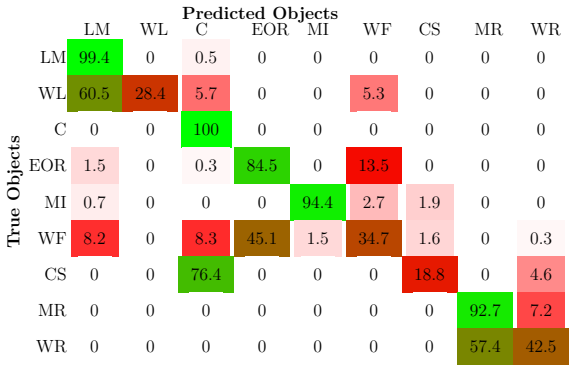


(c) Max 12 Batches

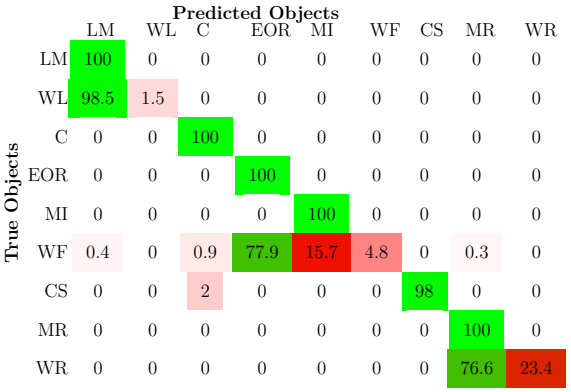
Figure A.3: Plots of the difference between clusters between batches with different handling of batch storage for the data file id29

A.2 Confusion Matrices

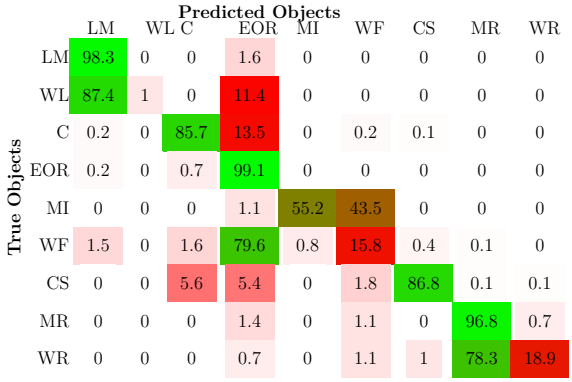
A.2.1 id5



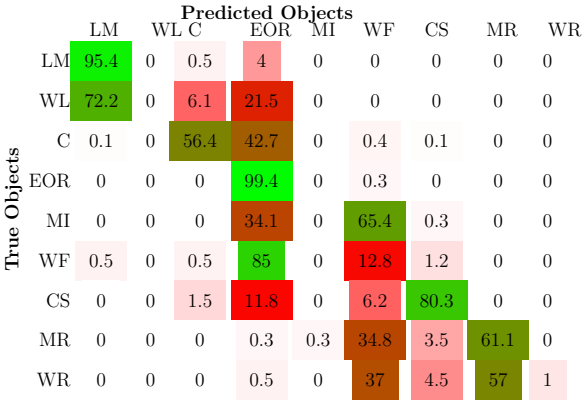
(a) EM algorithm on SEP data



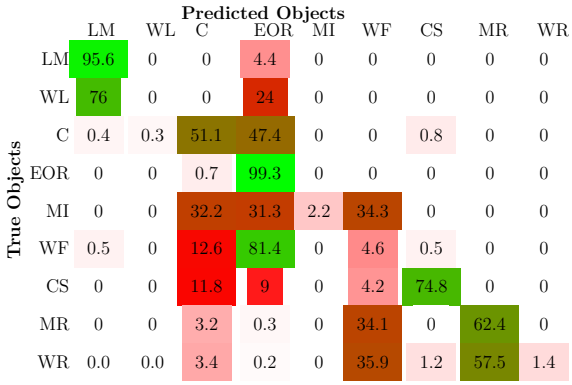
(b) GBG algorithm on SEP data



(c) GBG algorithm on TC data



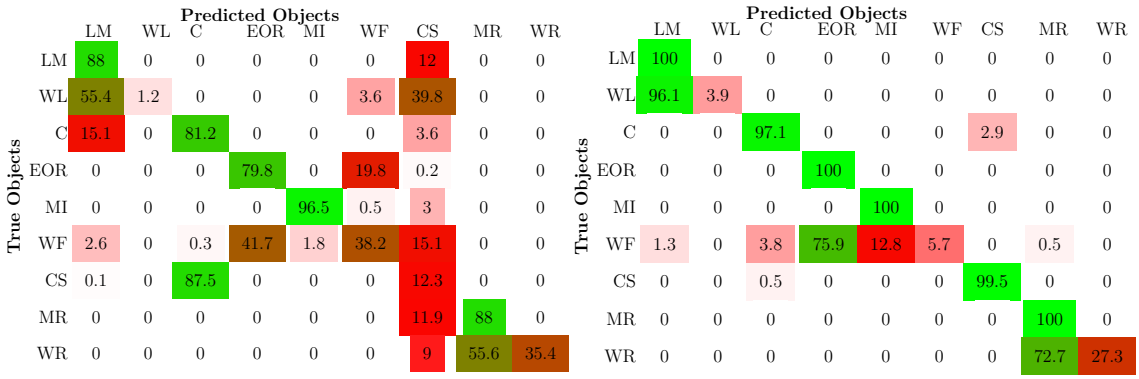
(d) Final algorithm on TC data



(e) Final algorithm on TC data using batches

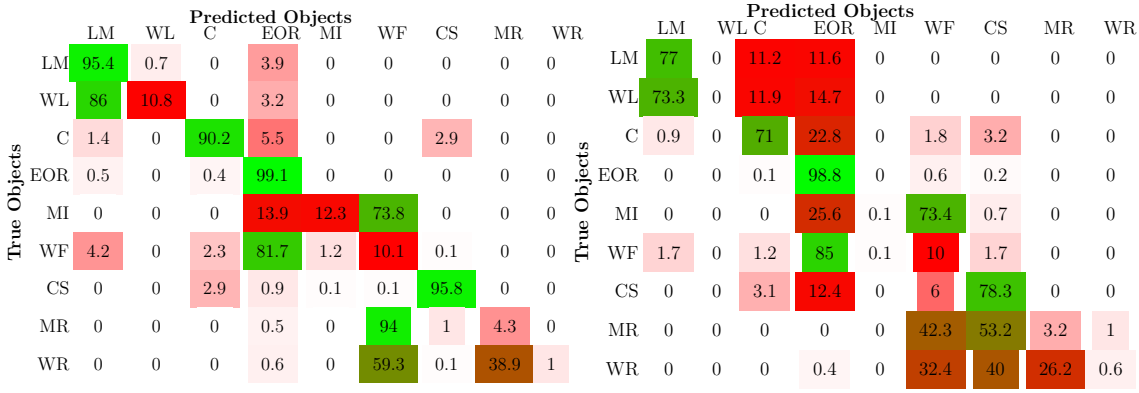
Figure A.4: Confusion matrices generated from the log file id5

A.2.2 id12



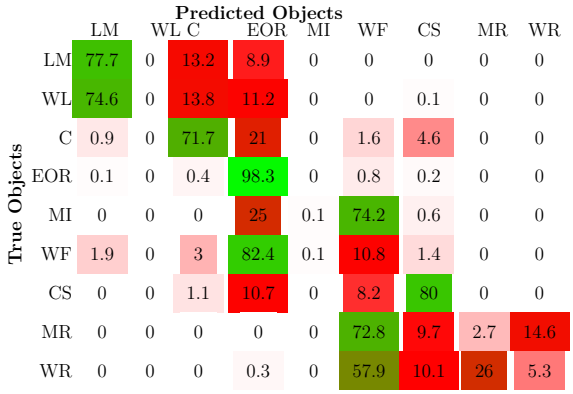
(a) EM algorithm on SEP data

(b) GBG algorithm on SEP data



(c) GBG algorithm on TC data

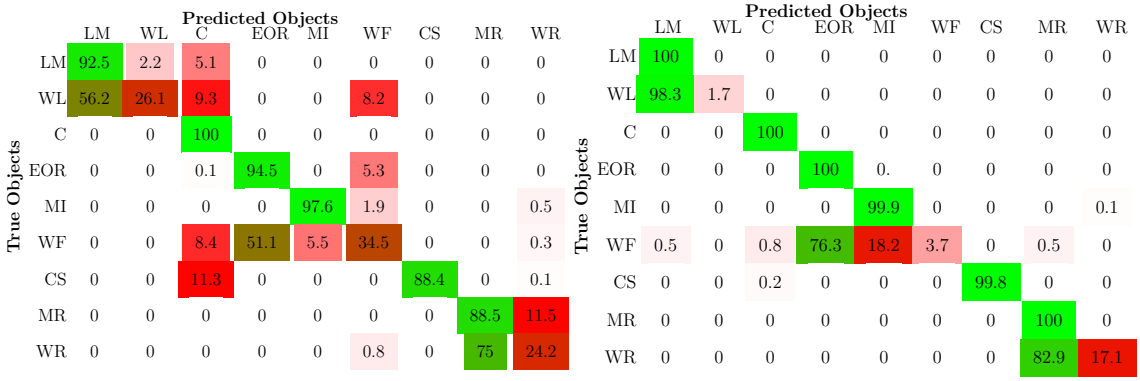
(d) Final algorithm on TC data



(e) Final algorithm on TC data using batches

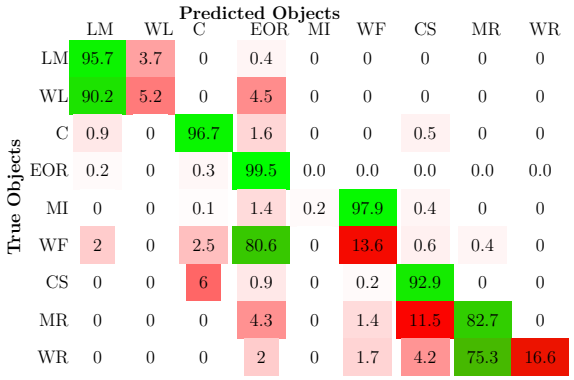
Figure A.5: Confusion matrices generated from the log file id12

A.2.3 id29

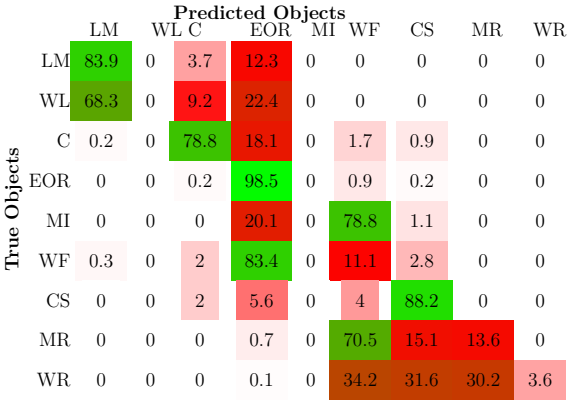


(a) EM algorithm on SEP data

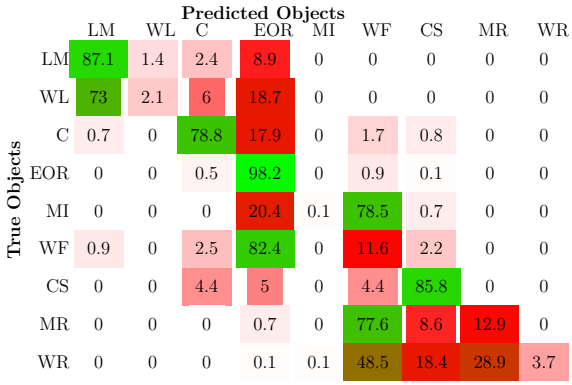
(b) GBG algorithm on SEP data



(c) GBG algorithm on TC data



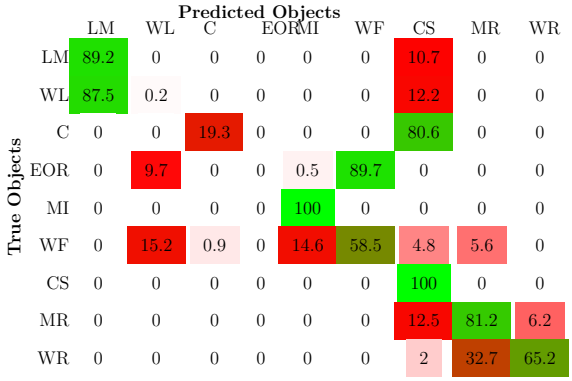
(d) Final algorithm on TC data



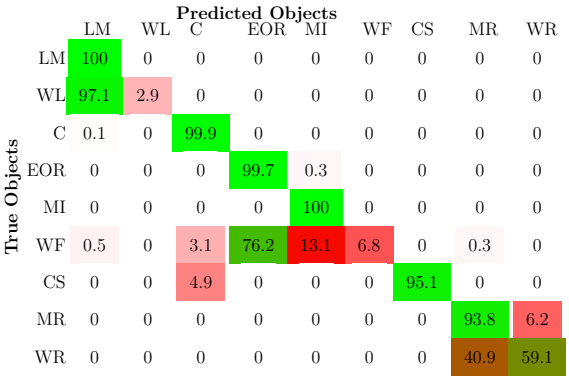
(e) Final algorithm on TC data using batches

Figure A.6: Confusion matrices generated from the log file id29

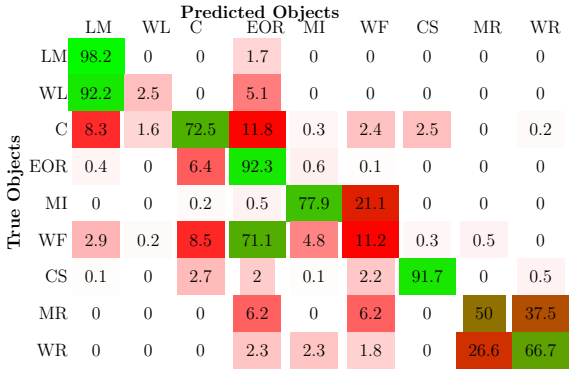
A.2.4 id64



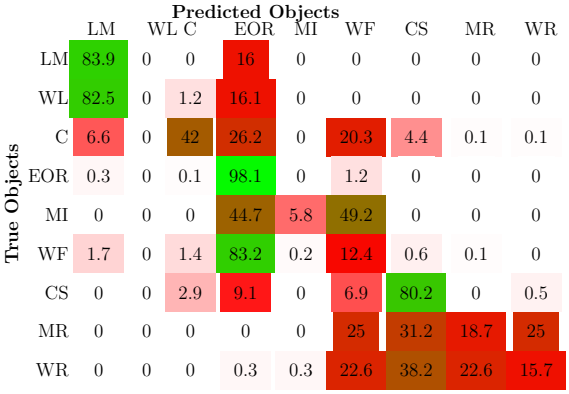
(a) EM algorithm on SEP data



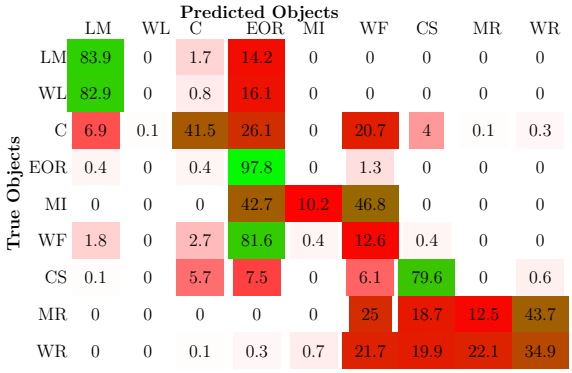
(b) GBG algorithm on SEP data



(c) GBG algorithm on TC data



(d) Final algorithm on TC data



(e) Final algorithm on TC data using batches

Figure A.7: Confusion matrices generated from the log file id64