4-14-2021 10:00 AM

Calibration Between Eye Tracker and Stereoscopic Vision System Employing a Linear Closed-Form Perspective-n-Point (PNP) Algorithm

Mohammad Karami, *The University of Western Ontario*

Supervisor: Dr. Steven Beauchemin, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

In many advanced driver assistance systems (ADAS) applications, it is essential to figure out where gaze of driver locates in image area of stereoscopic vision system. This problem, which involves a cross calibration between the stereo vision system and eye tracker, is a challenging task since the two systems are not consistent in modality and do not share a common image area. The eye tracker system provides a 3D gaze vector which describes the direction of driver's 3D line of gaze, while the stereoscopic vision system provides a depth image frame. In this thesis, this cross-calibration was performed with a closed-form solution that employs an efficient, linear time Perspective-n-Point algorithm. The main contribution of the present thesis is reformulation of this cross-calibration problem in a way that we would be able to employ PnP algorithm for providing a closed-form solution. The calibration process maps the 3D driver gaze vector into the surrounding outdoor environment. Moreover, the robustness of the algorithm with respect to noise is investigated on a set of synthetic data as well as in a lab-environment place.

Keywords: Cross-calibration, perspective-n-point, driver gaze mapping

Summary for Lay Audience

Advanced driver assistance systems (ADASs) have already been implemented in vehicles. Such examples are cruise control, warning systems for lane departure and automatic parking that have been recently introduced.

ADASs are supposed to provide a safe driving experience, but fatalities involving these systems increased during recent years. It is believed that if their full potential be realized, ADASs would have an annual benefit of around $800 billion by 2050 via mitigation in traffic congestion, energy consumption and road collisions. This goal cannot be achieved unless knowledge in ADASs is advanced to a greater extent.

ADASs are focused on perception of environments around the vehicles for assisting the driver, while little attention is given to the perception of driver behavior. It is believed that driver behavior can significantly improve ADASs as 95 percent of vehicle collision are due to human error. For detection of visual driver distraction, it is important to find out where the driver is looking. This is the main subject of current thesis.

Acknowlegements

I want to begin by expressing my sincere appreciation to Dr. Steven Beauchemin for his excellent supervision and feedback during the last two years. He gave me the time, resources and freedom to grow as a researcher. This work would not have been possible without his priceless guidance.

I would like to extend my thanks to my advisor Dr. Anwar Haque for his suggestions and guidance.

I would like also to thank Dr. Michael Bauer, Dr. Rahman Taufiq and Mr. Stephen Sweeny.

My officemates and friends have also been amazing during my study. Special thanks to Mohsen Shirpour, Farzan Heidari, Nima Khair Doost.

Last but not least, I would like to thank my family: my wife (Zahra Habibi), my parents (Fereshteh and Abdolhassan), and my sisters (Narges and Nastaran), for all their love and support.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

The government of Canada reported that in 2018 vehicle collisions resulted in almost 2000 fatalities and 152,000 injuries [14]. These catastrophes have led researchers to improve car safety by providing critical information to drivers, by way of Intelligent Transportation Systems.

Advanced driver assistance systems (ADASs) have already been implemented in vehicles. Such examples are cruise control, warning systems for lane departure and automatic parking.

The Society of Automotive Engineers defined five levels of driving automation [15]. Based on this categorization, Level 0 is the lowest level of driving automation and involves no automation. Level 1 involves partly automated driving support systems such as lane centering and adaptive cruise control, and level 2 involves breaking and acceleration support or collision-avoidance. In level 3, drivers are not responsible for controlling the vehicle, except in urgent situations. In level 4, the drivers are not responsible for driving except in limited conditions. Examples of

level 4 are local driverless taxis. For level 5, the vehicle is totally driverless and does not require a driver at any time.

ADASs are supposed to provide a safe driving experience, but fatalities involving these systems have increased during recent years [16]. It is believed that if their full potential would be realized, ADASs could have annual benefits of around $800 billion by 2050 via mitigation in traffic congestion, energy consumption and road collisions [17]. This goal cannot be achieved unless knowledge in ADASs is advanced to a greater extent. Such an unsolved ADASs problem is perception of the surrounding area which is critical for a safe navigation. Most of state-of-the-art computer vision algorithms that have been built during decades are still not robust to different lighting or weather conditions [18].

ADASs are focused on perception of environments around vehicles for assisting drivers, and they mainly rely on real-time computer-vision algorithms, supported by sensors. In lane departure warning, as one sample of ADAS, edge detection, image segmentation and feature tracking have been used successfully [19]. Thus, the robustness of computer vision algorithms has made driving automation at level 2 feasible. However, levels 3 and 4 of ADASs, where driver action is required in limited conditions, is challenging for several reasons.

First, the driver has to respond quickly when automation is switch off, leading to manual control requirements [20]. For high-level ADASs, drivers are engaged in non-driving related tasks (e.g. using mobile phones) which makes it hard to compute a sufficient transition time between automation and manual control. It is shown that the transition time is a critical factor that increase the risk of collision [21].

The second reason is that little attention is given to the perception of driver

behaviour. It is believed that understanding driver behaviour can significantly improve ADASs as 95 percent of vehicle collisions are due to human error [22]. For detection of visual driver distraction, eye as well as head movements are important factors to decide whether a warning should be given to the drivers [23]. However, it is challenging to perform experimental studies of driver behavior in real road conditions. The experimental setup is expensive and hard to design. The remote eye tracker itself can cost more than $30,000. It requires powerful systems for storage and data processing. Only one hour of driving can generate more than 250GB of data.

With the rise of deep learning and the advent of new sensors such as eye-trackers [16], ADASs have been improved [24]. Moreover, despite the significant number of ADAS studies, there is a lack in investigating the performance of driver monitoring systems in ADAS. This is partly due to the two reasons [18]. First, eye trackers cost is high and prohibitive to be implemented in the vehicles. However, it is expected that it will be commercialized at some point, and it is not going to be a problem in future. For instance, vehicle radars used to be around $15,000, but they are less than $100 nowadays. Second, the eye tracker modality modality is not consistent with the rest of vehicle sensors such as Lidar and stereo system. A robust ADAS should always perceive the driver attention and its surrounding environment, and this requires real-time data fusion from multiple sensors with the eye tracker that have different functionality, or the way it processes the input data. Considering different lighting and weather conditions, such as a robust state-of-the-art algorithm is not established yet [18, 25]. The data fusion, or calibration, between an eye tracker and a stereo vision system is the is the main subject of this thesis. Out of

cross-calibration, we would be able to perceive driver action and find out where the driver is looking, which is a crucial information for safe navigation.

Chapter 2

Literature survey

2.1 Overview

This Section provides a literature review on two aspects of eye tracking: (i) gaze estimation algorithms, and (ii) applications of gaze tracker in driver assistance systems.

2.2 Gaze estimation algorithms

Following Hansen's [1] survey, gaze estimation techniques can be categorized as 2D or 3D models. In 2D models, the input data is pupil image with glint, as shown in Figure 2.1, and the output of the model is a pixel coordinate on the screen. This mapping function can be done with regression [26] or neural network [2]. Schematic of the mapping function is displayed in Figure 2.2.

Since these models avoid explicitly calculating the intersection between the 3D gaze direction and 3D gazed objects, they are only used in vehicle simulator, and

not in real-road experiments [27]. Although 2D gaze models have their own challenges, such as recognition of compatible input gaze data with the calibration data [28], they are heavily studied and most of the challenges are remained for 3D gaze models.

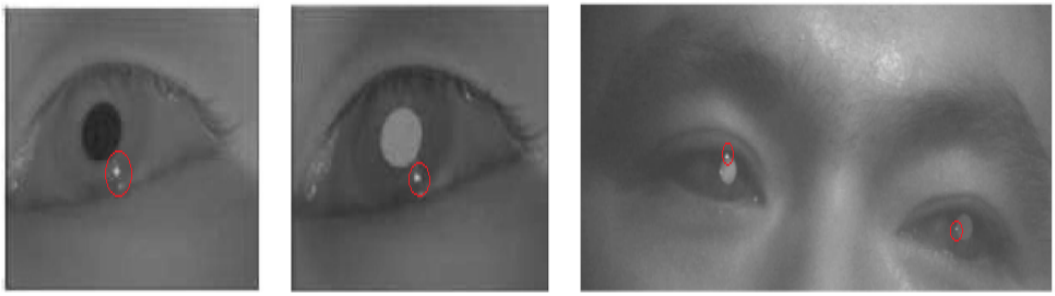


Figure 2.1: Different pupil images (a) dark, (b) bright, and (c) bright with different intensity. The small reflection on the cornea surface in the dark and bright pupil images is referred to as the glint, shown as red circles. Obtained with modifications from Ref. [1]

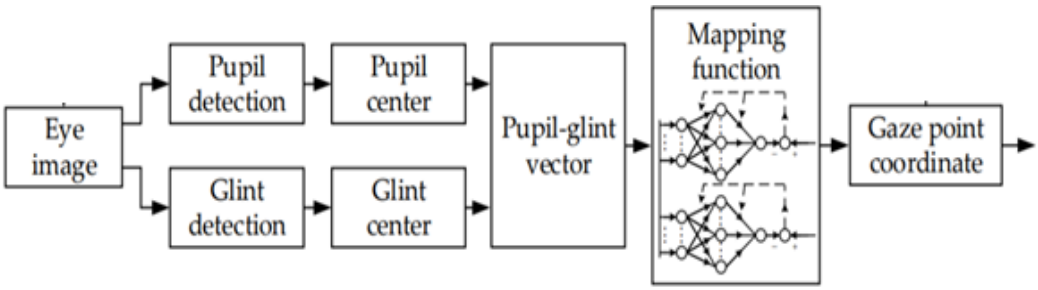


Figure 2.2: Schematic of the mapping function in 2D gaze estimation models. Obtained with modifications from Ref. [2]

In 3D gaze estimation algorithms, 3D human eye ball is modelled as a sphere with a constant radius, and the difference between 3D position of iris center and the position of eye ball center defines the 3D gaze vector [3], displayed schemat-

ically in Figure 2.3. The main challenge, herein, is detection of the position of iris center [1]. Due to this reason, we have two types of gaze trackers: Remote and head-mounted. Remote eye trackers are considered as non-intrusive monitoring technique (i.e. non-restriction technique), and consequently more expensive than head-mounted eye trackers. In this thesis, a commercial remote eye tracker, manufactured by FaceLab company, has been used for tracking gaze data, shown in Figure 3.1.

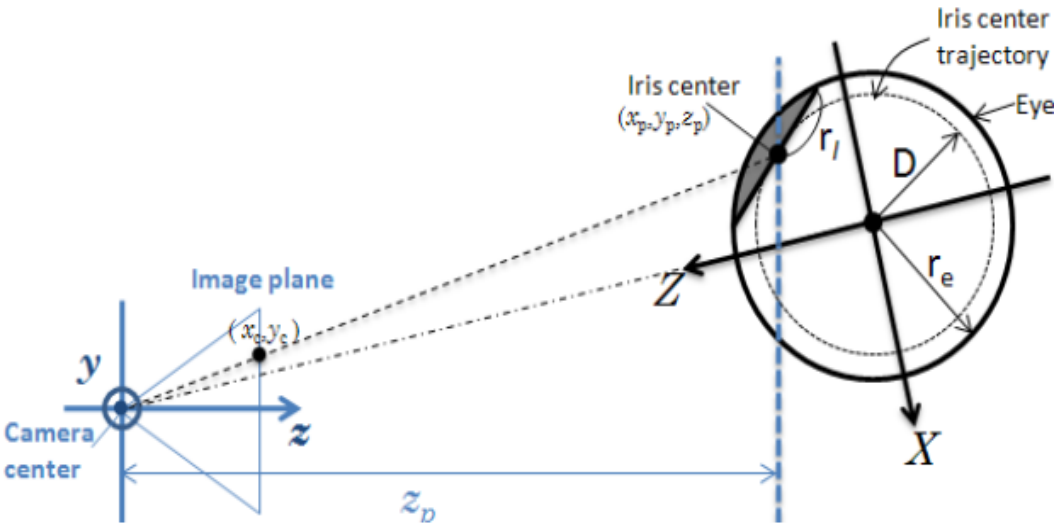


Figure 2.3: Schematic of the 3D gaze estimation models. Obtained with modifications from Ref. [3]

To figure out the location where the driver is looking, we need to map the 3D driver gaze vector in image frame provided from a stereo vision system. This process, refers to as cross-calibration, requires depth information of the surrounding objects, which is provided from a stereo vision system. Based on the 3D gaze vector provided from the eye tracker and depth information provided from stereoscopic vision system, we would be able to recognize the gaze target as an intersection of

the gaze vector with the surrounding objects [1].

This calibration process is a challenging task as there is no common imaging area between the two systems and the two systems process different types of data. Eye tracker processes a 3D vector while stereo vision system processes RGB-Depth data, i.e. inconsistent modalities. There are very few contributions on this cross-calibration problem. Takagi et. al. [4] simplified the cross-calibration by assuming that one axis of coordinate systems of eye tracker and stereoscopic vision are parallel. The relationship between each camera is shown in Figure 2.4. This assumption constrained them to implement the algorithm in a vehicle simulator. Hennessey and Lawrence presented a 3D PoG (Point of Gaze) method which employs eye vergence to estimate the 3D position of a fixated object [29]. Recently, Kang et. al. [5] and Kowsari et. al. [30] proposed an iterative algorithm for the calibration problem with experimental setup shown in Figure 2.5. They provided a cost function to minimize the convergence error.

The above-literature review shows a lack of robust cross-calibration between eye trackers and stereoscopic vision systems. Such a lack of robust algorithm with easy implementation has led researchers to use virtual simulators or use iterative algorithms, whose solution depends on the initialization. To the best of our knowledge, the proposed algorithm presented in the current thesis is the first closed form solution to the problem, which makes it easy and robust to implement. In the next Section, applications of the cross-calibration problem in driver assistance systems are reviewed.

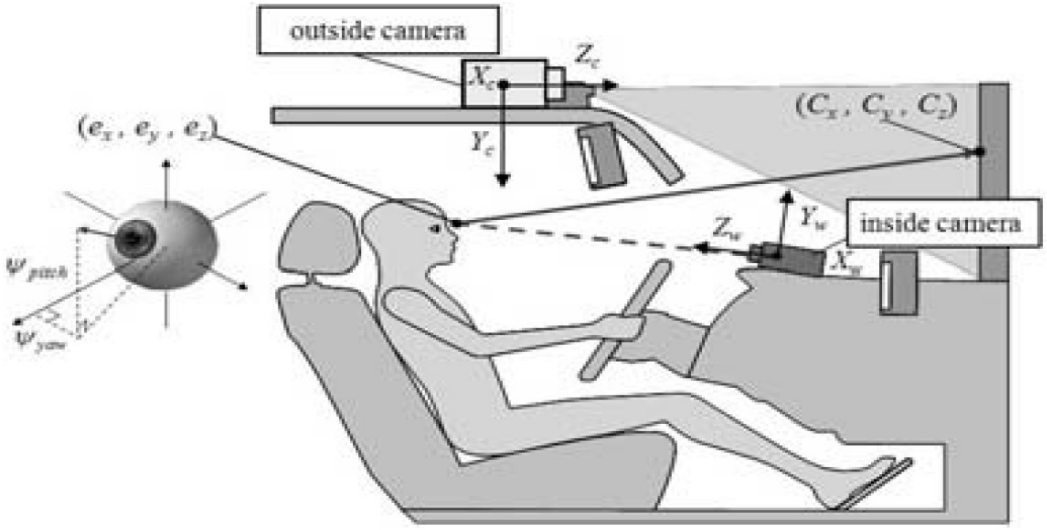


Figure 2.4: Relationship between each gaze tracker and stereo system, studied in Ref. [4]

2.3 Applications of gaze tracker in driver assistance systems

Lethaus and Rataj [31] tried to find a reflection between eye movement and driver action. This relationship has been experimented by Land and Horwood [32] as well as Land [33]. They showed that a time lag of 0.8 to 1 s exists between eye movements and steering maneuvers, suggesting that the driver gaze contains important information for action prediction.

Lethaus et. al. [6] used supervised learning techniques on gaze data to derive information about a driver's next plan. They showed that gaze data is a 'stand-alone' predictive action of lane change with a time lag of 10 s, as shown in Figure 2.6.

Li et. al. [7] predict to find driver's lane-change intent by considering the

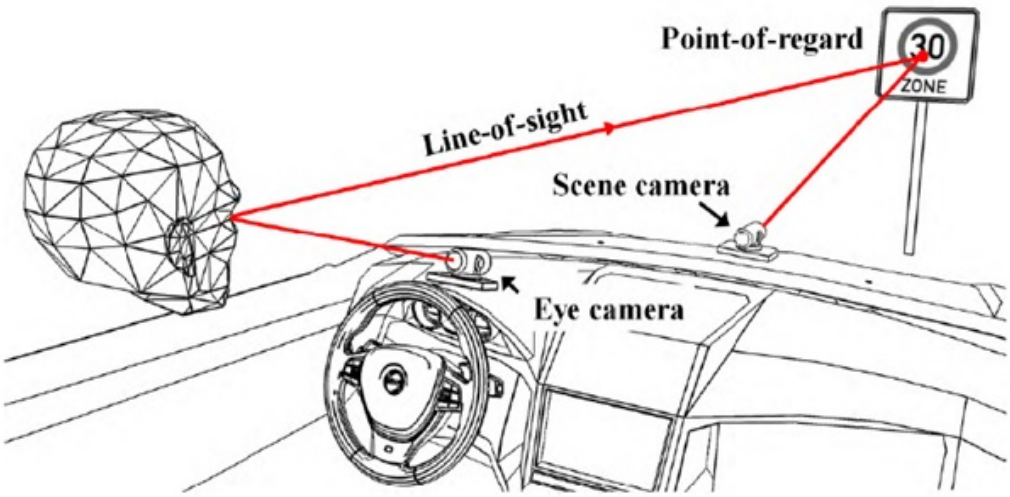


Figure 2.5: Schematic setup of gaze tracker and stereo system studied in Ref. [5]

surrounding traffic. They used a driving simulator (Figure 2.7) with three different scenarios, as shown in Figure 2.8. Using driver gaze data, they proposed a Bayesian network-based model that estimates the driver's intent in 4.5 seconds ahead with a better accuracy than Support Vector Machine (SVM) model.

Mori et. al. [8] studied real-world driving instead of vehicle simulators and found a relationship between gaze behaviour and risks posed by surrounding vehicles. They divided the surrounding area of a vehicle into eight zones (Figure 2.9) and defined risk as time to collision. They showed that gaze behaviour is indicative of drivers awareness levels.

Pech et. al. [9] analysed driver gaze and head pose before lane change to extract and estimate driver behaviours presenting its intent. They defined six different gaze areas: front window ahead (FW), left mirror (LM), right mirror (RM), rear

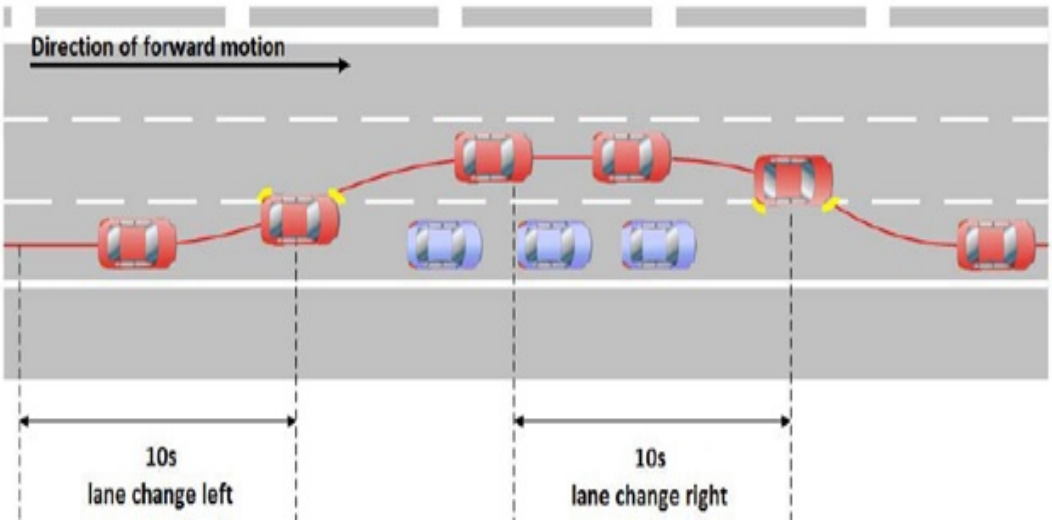


Figure 2.6: Schematic relationship between lane change action and gaze data pattern, adapted from Ref. [6]

view mirror (RVM), left window (LW) and right window (RW). The vehicle was equipped with six cameras as shown in Figure 2.10. They found interesting patterns of side-to-side head rotations, as shown in Figure 2.11. However, due to the absence of driver gaze mapping on a stereo image frame, it was challenging to distinguish the nearby defined gaze areas, such as right mirror (RM) and right window (RW). This issue resulted in low classification result of left window (LW) and right window (RW), presented in Table 2.1. Meanwhile, they had to use six cameras for tracking driver in the defined gaze areas, while this would not be an issue if the driver gaze mapping was available.

Diederichs and Spath [10] used driver gaze data for developing an automated braking system for pedestrian protection. They considered different scenarios for pedestrian movement for data collection. As shown in Figure 2.12, the pedestrian is either walking along the road or crossing the road. They tracked the distance

Figure 2.7: A screen shot of driving simulator used in the study of Ref. [7]

between eye fixation and center of the pedestrian, shown in Figure 2.13. However, they had to use a driving simulator in their study due to the absence of driver gaze mapping.

Ahlstrom et. al. [11] performed the first study on a real time distraction warning system based on driver gaze data. They carried out experiments with seven drivers who drove an average of 4351 kilometers on real road conditions. They defined a field view relevant for driving as a cone of 90^0 (Figure 2.14), and any deviation out of this field of view for more than 2 seconds is considered as driver distraction. To figure out where the drivers are looking when their gaze of line is deviated from the field of view, they divided the vehicle into different area zones based on the assumption that the drivers are supposed to look at them during standard driving tasks, see Figure 2.15. The gaze areas were the windscreen, the right front window, the left front window, the right rear view mirror, the left rear view mirror, the centre

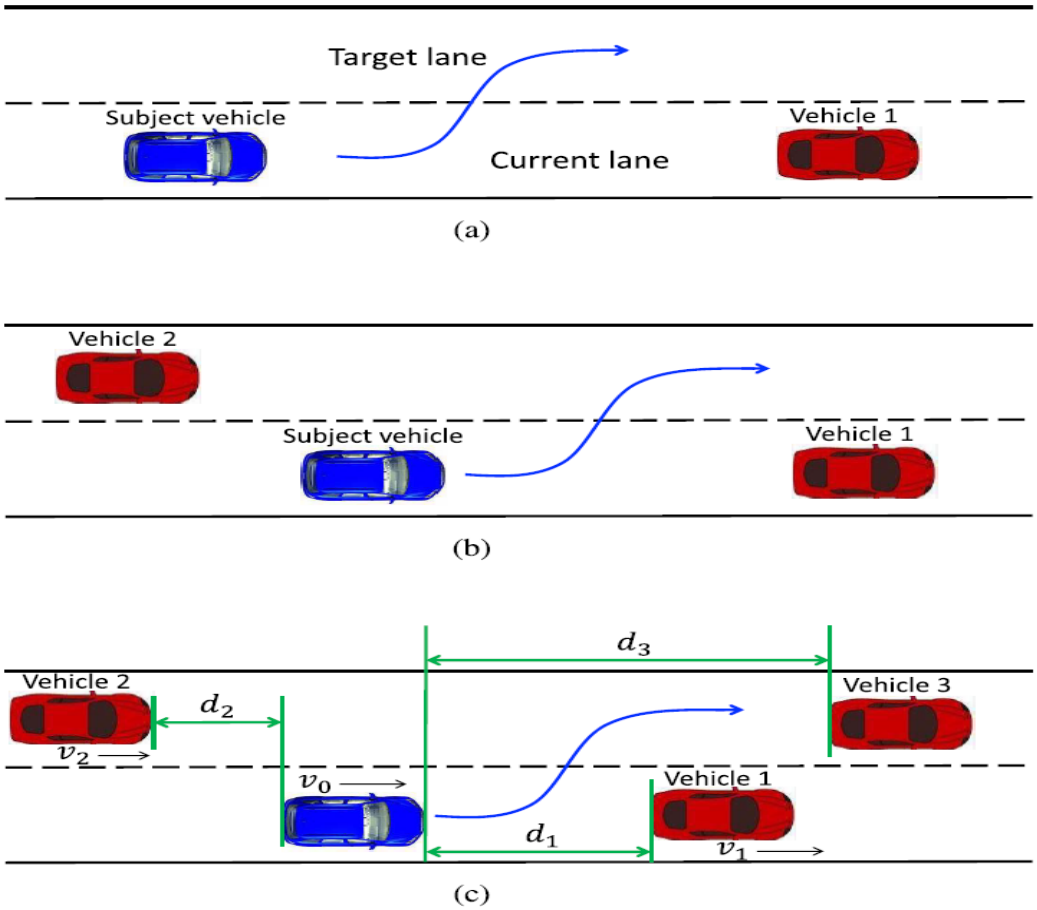


Figure 2.8: Three different scenarios considered for lane-change driver's intent in Ref. [7]

rear view mirror, the dashboard, the speedometer, the middle console, the glove box, the left front door, the right front door, the floor of the car, the foot area and the roof. They showed that the suggested warning system changes the driver visual behaviour in the desired direction. However, there were some limitations on their work: the field view was static and not changed dramatically with upcoming road geometry, and the driver's intent was not taken into consideration.

Zabihi et. al. [13] used 3D gaze data to detect vehicles that are most likely the

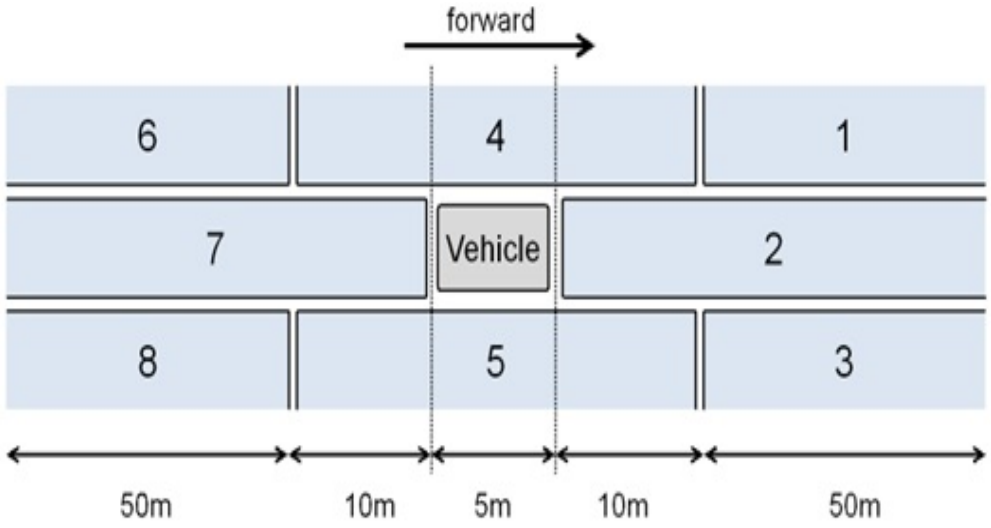


Figure 2.9: Divided surrounding area of vehicle for evaluation of risk, adapted from Ref. [8]

driver was aware of. They defined an attentional gaze cone whose radius depended on the distance between the fixated object and the driver (Figure 2.16). The cone angle was set to 6.50^0 Ref. [4]. Following the definition of the gaze cone, they managed to detect the vehicles that elicited ocular responses from drivers. Figure 2.17 shows a sample of detection results. This study was performed only for vehicles, but could be easily extended to other types of objects such as pedestrians.

In summary, the above-literature review shows that the lack of presence of robust cross-calibration algorithm has been a constraint on ADAS researchers. Mori et al. [8] looked into the relation between gaze behaviour and driver awareness. They did experiment in real road conditions, but they simplified the problem into a classification problem by dividing vehicle surrounding areas into 8 different zones. Moreover, Kircher et al. [11] suggested a gaze-based driver distraction warning system by defining a 90^0 cone view. They did experiment in a real road condition.

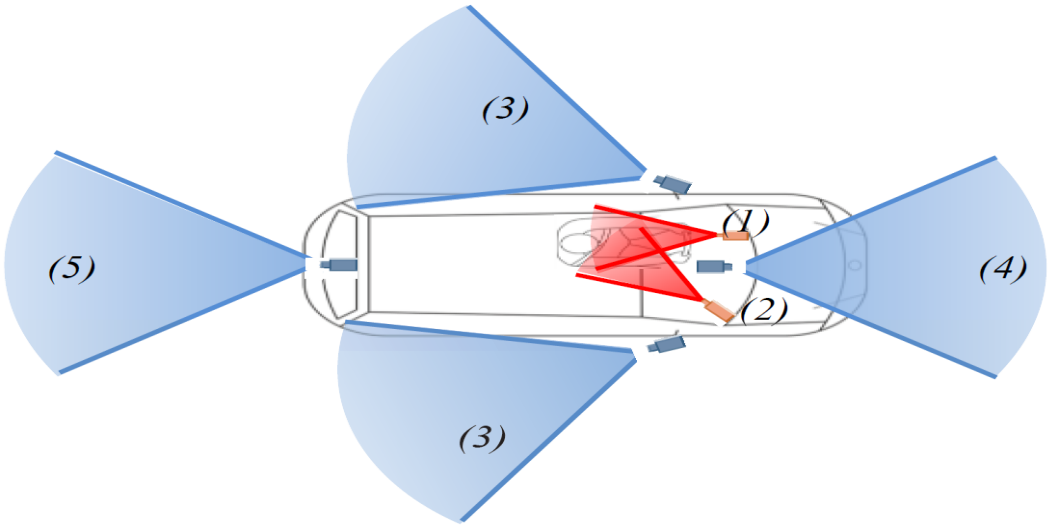


Figure 2.10: Schematic configuration of cameras inside the vehicle for the study of gaze/head behaviour, adapted from [9]

However, due to the lack of driver gaze mapping information, the cone view was static and not dynamic by implementing the surrounding areas information. Due to the challenges in real road experiments, some ADAS researchers have used driver simulator, in which 2D gaze estimation with a 2D target plane is required [34]. For instance Lie et al.[7] used a driver simulator to find the relation between lane change action and eye movement. However, vehicle simulators are far away from the real conditions and the real-road experiments.

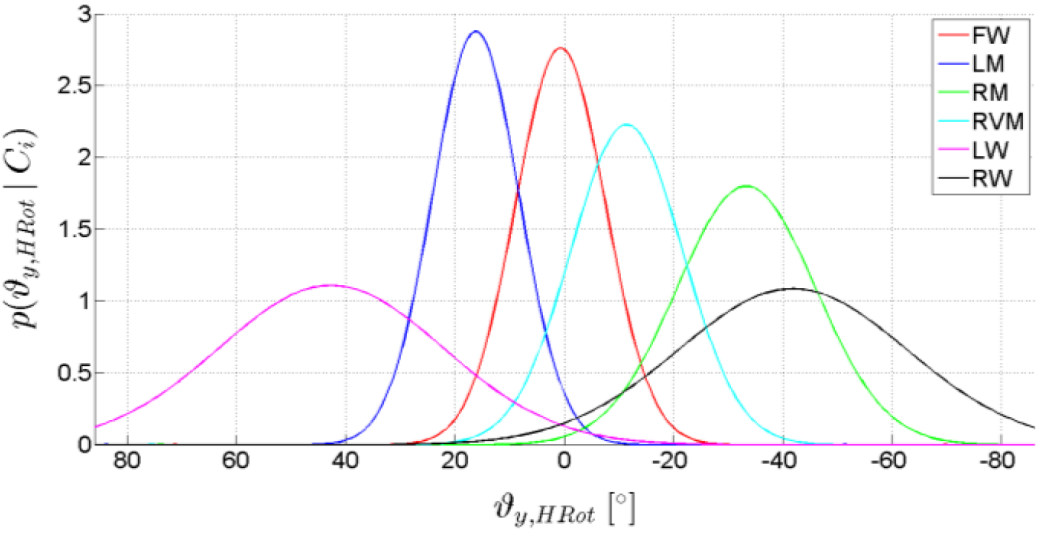


Figure 2.11: Class distribution of the six different gaze areas versus side-to-side head rotation, adapted from [9].

Table 2.1: Classification result of the Ref. [9]

Glance area	Positive predictive value
All	0.81
FW	0.96
LM	0.86
RM	0.75
RVM	0.86
LW	0.06
RW	-

Figure 2.12: Two different pedestrian walking scenarios used in the driving simulator [10]. Black arrow shows the pedestrian path.

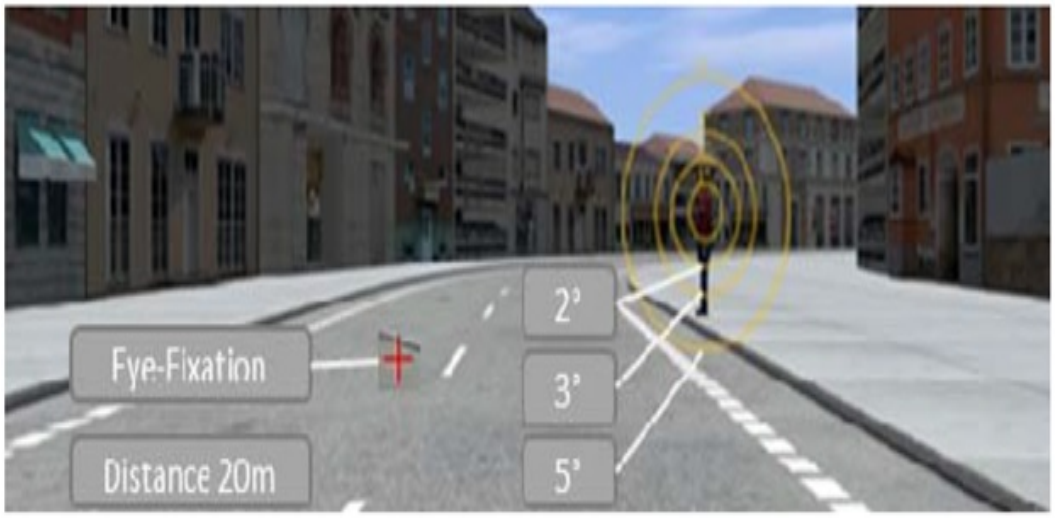


Figure 2.13: Eye fixation and its distance from pedestrian center [10].

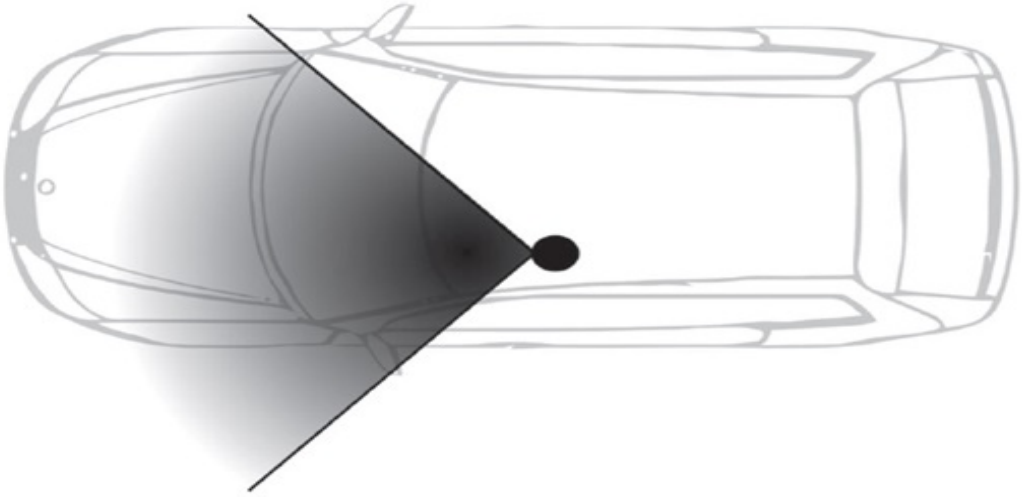


Figure 2.14: Driving field relevant view, defined in [11]. The mirrors are not part of this field of view.

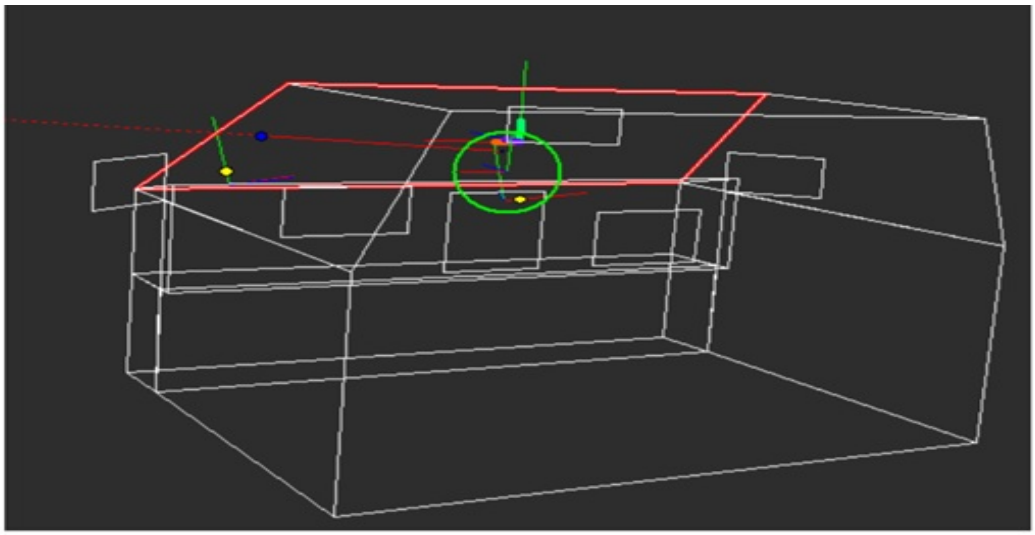


Figure 2.15: The defined gaze areas in [11]. The white rectangles show the gaze areas, and the green circle shows the head position. The red line emanating from the head is gaze direction. [12].

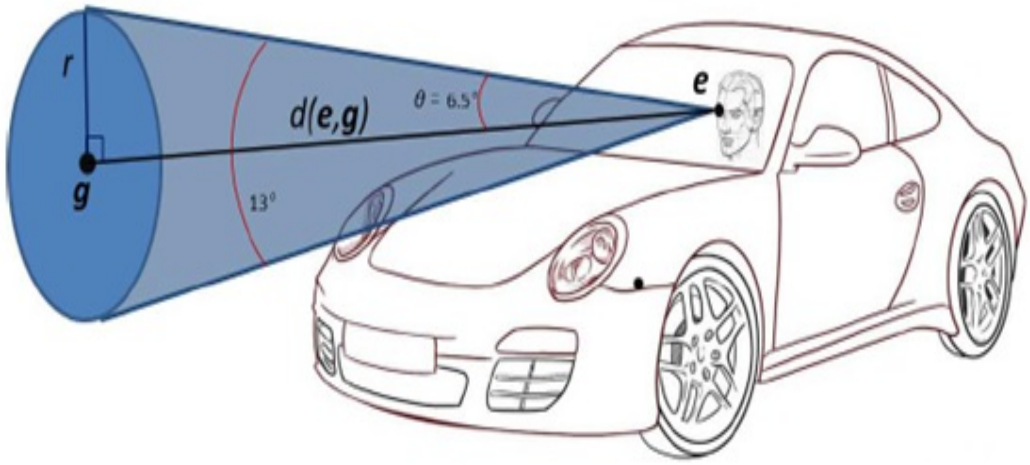


Figure 2.16: Driver attentional gaze cone defined in [13].

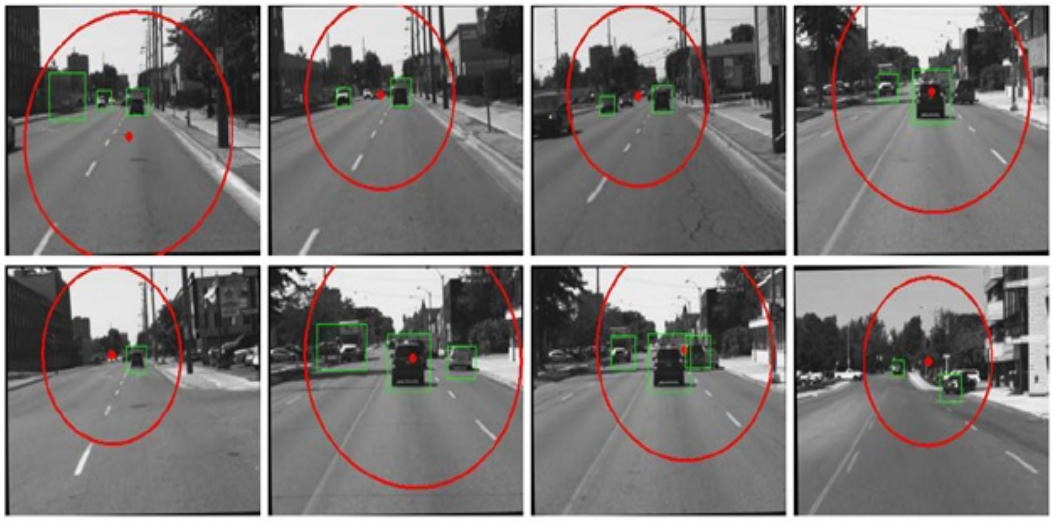


Figure 2.17: Detection of vehicles that fall in the gaze cone, from [13].

Chapter 3

Cross-calibration algorithm and error analysis

3.1 Problem overview

The goal of this thesis is the cross-calibration process between a FaceLab eye tracker [35], facing the driver and a RealSense stereoscopic vision system [36] facing the surroundings. The eye tracker provides a 3D gaze vector of one of the driver's eye, $\vec{g} = (g_x, g_y, g_z)$ and the stereoscopic vision system provides RGB-Depth image frames.

The physical configurations of the FaceLab eye tracker and RealSense stereoscopic vision system are shown in Figure 3.1. The FaceLab eye tracker is a commercialized product that is experimentally designed for monitoring driver action in a remote and non-intrusive manner. It has its own machine and needs to be calibrated for each individual driver. Its operating system is Windows 7 and requires

socket programming to retrieve its gaze data through a LAN cable in real-time. For more information about this process see the Appendix A. The stereoscopic vision system provides RGB-Depth as well as 3D point cloud of the surrounding objects. Its depth range is around 6 meter with 30 HZ FPS. Its data can be retrieved with USB 3.0 cable.

The cross-calibration process between the eye tracker and the stereoscopic vision system refers to the process of mapping 3D gaze vector on the image frame. To achieve this, we need to find rotation and translation matrices that transforms the driver's gaze from coordinate system of the eye tracker to coordinate system of the stereoscopic vision system. Herein, estimation of the translation and rotation matrices has been done with the help of Perspective-n-Point (PnP) algorithm. However, it should be noted PnP algorithm has been designed for a totally different set of problems, and thus the present cross-calibration problem needs to be re-formulated in order to be solved with PnP algorithm. In the following sections, first, the PnP algorithm is described, and then re-formulation of the cross-calibration problem will be explained.

3.2 PnP algorithm

The formulation of the PnP algorithm has been shown schematically in Figure 3.2. The PnP algorithm finds the pose of a camera using a set of object points, shown in orange color, in the world coordinate system and their corresponding projection, shown in yellow color, on the image frame. The PnP algorithm returns the rotation and translation matrices that project object points into the image frame.

PnP problems can be solved with two methods. The first method is P3P. The minimal PnP problem with a finite number of solutions require three point correspondences ($n = 3$). P3P methods are not usually robust to noises, although the latest P3P algorithm by Persson et al. [38] in 2018 has been found promising. The second method solves the general problem of PnP for $n \geq 4$, to deal with outliers in the data. Table 3.1 summarizes some of the well-known closed form approaches to the general case of PnP problems.

Herein, the algorithm proposed by Lepetit et. al. [39] is chosen since it is available in OpenCV library, resulting in easy implementation. The central idea in Lepetit et al. [39] algorithm is expressing the n 3D points as a weighted sum of four virtual control points. This expression reduces the PnP problem into the estimation of the coordinates of these control points, which can be obtained in non-iterative manner. The time complexity of Lepetit et. al. [39] is linear $O(n)$, which is in contrast to previous methods that are in $O(n^2)$ [40], $O(n^5)$ [41], and $O(n^8)$ [42].

Table 3.1: Time complexity of recent closed-form PnP algorithms.

Authors	Year	Time Complexity
Quan and Lan [41]	1999	$O(n^5)$
Ansar and Daniilidis [42]	2003	$O(n^8)$
Fiore [40]	2001	$O(n^2)$
Lepetit et al. [39]	2009	$O(n)$

PnP algorithm requires a set of correspondences between 3D points, P_i in the world coordinate system and their 2D points, p_i onto the image frame. It also assumes that the intrinsic camera matrix, which consists of focal length and optical center, is known to us in prior. Using these data as input, PnP returns the rotation and translation matrices that project object points into the image frame. The PnP

algorithm involves with solving the following equation for obtaining the rotation and translation matrices $[R|t]$:

$$S_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{3.1}$$

which can be also written as:

$$S_i p_i = M_c M_t P_i \tag{3.2}$$

Where (X_i, Y_i, Z_i) or P_i are the object points expressed in the world coordinate system. (x_i, y_i) or p_i are the projections of object points on the image frame. S_i is the scaling factor, and M_c is the intrinsic camera matrix. PnP solves the equation to obtain the R and t matrices.

Briefly, PnP algorithm is designed to estimate the rotation and translation matrices that project the object points onto the image frame. This problem, which refers to as camera pose estimation, is completely different from the present cross-calibration between eye tracker and stereo vision system. In order to be able to use PnP algorithm for the present cross-calibration problem, we need to re-formulate the cross-calibration, which is explained in the next section.

3.3 Cross-calibration algorithm

In the present cross-calibration problem, we want to find out rotation and translation matrices that project the 3D gaze vector of one of the driver's eye, $\vec{g} = (g_x, g_y, g_z)$, on the image frame of the stereoscopic vision system. However, PnP algorithm requires a set of correspondences between 3D points and their 2D point, and it has not been designed to deal with gaze vectors. To solve this issue, we need to convert the 3D gaze vectors into 2D image points. For the purpose of conversion, we assume that the head position remains constant and a virtual plane is located at a distance of 1 meter and perpendicular to the line of sight of the eye tracker. In other words, we have created a virtual camera in which the virtual plane is assumed as an image frame and the gaze vectors are originating from the optical center of the eye tracker. It should be also noted that the distance of the virtual plane from driver's head is arbitrary since it is only associated to the values of intrinsic virtual camera matrix.

The construction of this virtual camera through the assumption of the virtual plane can help us to convert each of the 3D gaze vectors into the 2D image points, required by the PnP algorithm. To do this conversion, we find the intersection of the gaze points with the virtual plane, and these points of intersections are the 2D image points, needed for the PnP algorithm. Since we assumed that the virtual plane is located at a distance of 1 meter, the optical center and the focal length of the virtual camera are: $(c_x, c_y) = (0, 0)$ and $(f_x, f_y) = (1, 1)$. These parameters are required by PnP algorithm for formation of the intrinsic virtual camera matrix. The schematic of the virtual plane configuration is shown in Figure 3.3.

It should be noted that the construction of the virtual camera for solving the

cross-calibration problem will put a limitation on the implementation of the algorithm. This limitation is requirement of a constant and a fixed head position. This means that any head movement is considered as a noise in estimation of the rotation and translation matrices. However, the effects of the noise would be reduced, when the 3D object points (or the point where the driver is looking) is farther away from the stereo vision system. Herein, these 3D object points are referred to as salient points.

The whole process of the cross-calibration problem which employs PnP algorithm is summarized in Algorithm 1. The input to the algorithm is the virtual camera matrix, M_c, which consists of optical center and the focal length of the virtual camera. Herein, the distance of the virtual plane is 1 meter from the driver's head, and thus the camera matrix is an identity matrix. The output of the algorithm are the rotation and translation matrices that we can project the 3D gaze vector on the image frame. The algorithm also returns the eye position, e_i, to monitor the noise introduced from deviation of the head position from the initial position.

It is worth mentioning that during the calibration process, we compute the set of the salient points from the image frame provided by the stereo vision system using the function GoodFeaturesToTrack from OpenCV library. These salient points, P_i, are displayed by blue circle in Figure 3.5. Afterwards, we ask the driver to fixate its gaze on each salient point for two seconds, during which gaze data from the eye tracker and its correspondence 3D salient point from stereo vision system are collected.

Algorithm 1 Cross-calibration algorithm

INPUT: Virtual Camera Matrix M_c
OUTPUT: Extrinsic Calibration Parameters $[R|t]$ and eye position e

Select $n \geq 4$ 3D salient points P_i from RGB-Depth vision system
for all i in n **do**
  Require the observer to fixate its gaze on the 3D salient point P_i in the environment
  Collect gaze vector, $\hat{g}_i$, and the eye position, $\hat{e}_i$, from the eye tracker
  $\vec{g}$ = average of sample $\hat{g}_i$
  e_i = average of sample $\hat{e}_i$
  p_i = point of intersection of the $\vec{g}$ with the virtual plane
end for
Compute PnP(M_c, p_i, P_i)
Return $[R|t]$, e_i

3.4 Error analysis

Two important aspects of the algorithm need to be studied: i) its accuracy and ii) its robustness toward noise. For this purpose, we performed error analysis on two distinct data sets. The first one is a set of synthetic data without noise, while the other data set is from a lab environment. Note that one of the main sources of noises, herein, can be attributed to the head movement, violating the assumption that gaze vectors origination from the optical center of the eye tracker.

3.4.1 Synthetic data

The synthetic data set in this Section is primarily used for the evaluation of the accuracy of the algorithm and its noise robustness. The synthetic data set is provided in Table 3.2.

To study the robustness of the algorithm to noise, we added zero-mean Gaussian

noise to the gaze vector, for which the value of the standard deviation (σ) of the Gaussian noise increases. It should be noted that the type of noise in real condition is unknown to us, and thus, herein, a Gaussian noise has been considered due to its simplicity. This process is summarized in Algorithm 2.

Herein, we applied two techniques of *PnPRansac* provided from the OpenCV library: (i) Levenberg-Marquardt method and (ii) Lepetit et al. [39] algorithm, referred to as efficient PnP. Levenberg-Marquardt algorithm is an iterative technique which tries to minimize the reprojection error, while the efficient PnP algorithm provides a linear closed-form solution to the PnP problem. This comparison provides us additional information needed for the solution to the calibration problem.

For the set of synthetic data set without noise, Table 3.2, both algorithms provide similar results with a reprojection error of 0.35 mm, which is very close to zero. This small value of the reprojection error validates the correctness of the cross-calibration algorithm. When we add noise into the synthetic data set, a difference between the two algorithms emerges (see Figure 3.4). It can be seen that the reprojection error increases almost linearly with amplifying noise, suggesting a linear relationship between noise magnitude and reprojection error. Moreover, it shows that both algorithms have a comparative noise robustness, but, herein, EPnP has been selected for employment in the cross-calibration algorithm since it provides a closed-form solution.

3.4.2 Lab-environment experiment

In this Section, we apply the algorithm in a lab environment. Figure 3.5 shows the image frame with its salient points, displayed in blue points, and its depth map

Algorithm 2 Noise robustness algorithm

INPUT: Lists *gazes*, *salients* and *CameraMatrix*, see Algorithm 1
OUTPUT: Reprojection error
For concatenation of reprojection errors
$errors$ = [ ]
For concatenation of projected noisy gazes
p_n = [ ]
for $k = 1$ *to* 20 **do**
  for all g_i in *gazes* **do**
    m = magnitude of g_i
    $\sigma = 0.01 * k * m$
    for $i = 1$ *to* 3 **do**
      $random\_value(i)$ = a Gaussian random quantity with zero-mean and standard deviation σ
    end for
    $noisy\_gaze$ = add $random\_value(i)$ to each component of g_i
    $p_{k,n}$ = projection of $noisy\_gaze$ on the virtual plane
    # concatenate $p_{k,n}$ into the list p_n
    p_n.append($p_{k,n}$)
  end for
  # compute R, T matrices from the noisy projection points and noiseless 3D salient points
  $R, T = PnP(p_n, salients, CameraMatrix)$
  # obtain a set of 3D salient points, using p_n, R, T
  $P_n = R^{-1}{M_c}^{-1}S_i(p_n) - R^{-1}T$
  $reprojection\_error$ = average squared difference between P_n and $salients$
  # concatenate $reprojection\_error$ into the list $errors$
  $errors$.append($reprojection\_error$)
end for

RETURN $errors$

Table 3.2: Synthetic data set of 3D salient points and gaze vectors.

3D salient point(mm) (X, Y, Z)	Gaze Vector $\vec{g} = (g_x, g_y, g_z)$
(0, 3, 50)	(0, 3, 30)
(2, -5, 47)	(-3, -5, 28)
(-1, 7, 60)	(10, 7, 31)
(5, -1, 40)	(-10, -1, 25)
(0, 2, 45)	(-5, 2, 30)
(3, -4, 44)	(-6, -4, 27)

frame. For the first case, the driver head or eye location remains fixed during the whole cross-calibration process. The experimental data set is provided in Table 3.3. Note that the small variations in eye locations in the data set is due to the fact that head movement can be reduced but cannot be equal to zero.

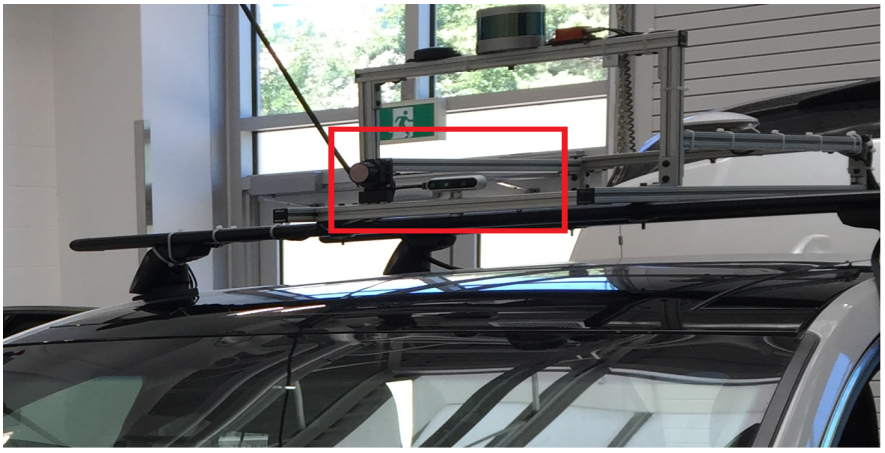
Applying PnPRansac from the OpenCV 4.2.0 library provides with the rotation and translation matrices with a reprojection error of 78.26 mm. Increasing the number of salient points to 19 points in another experiment did not affect the reprojection error, and it was almost equal to 110 mm, respectively.

The present algorithm error is in the range of 1.3% to 3.3% with respect to the depth of salient points. This normalized value of error is obtained by dividing the value of reprojection error, 77 mm, by the depth of salient points, value of Z presented in the Table 3.3. This normalized value of error for the previous studies with iterative algorithms is 8.9% in [29], 6.6% in [4] and 2.1% in [5]. Moreover, the time latency of present algorithm is in the order of milliseconds, while this value is 1.5 seconds for the algorithm suggested in [29] due to its image preprocessing steps. It should be noted that the present closed-form PnP algorithm has linear time complexity $O(n)$, which is in contrast to the previous iterative algorithms with the

following time complexity: $O(T(n^3))$ in [30] and $O(T(n^2))$ in [4], where T is the number of iterations and n is the number of salient points.

Table 3.3: Experimental data set.

3D salient point(mm) (X, Y, Z)	Gaze Vector $\vec{g} = (g_x, g_y, g_z)$	Eye Location(mm)) (X_e, Y_e, Z_e)
(411.83,111.53,3145.0)	(-0.11,-0.04,0.99)	(43.47,85.07,809.82)
(-509.04,875.49,3066.0)	(0.13,0.18,0.97)	(42.80,84.92,811.13)
(-903.66,426.11,3586.0)	(0.18,0.03,0.98)	(42.60,85.14,811.79)
(645.91,1133.50,3051.0)	(-0.12,0.23,0.96)	(42.03,86.14,813.05)
(2.26,1152.40,3262.0)	(0.00,0.21,0.97)	(40.56,85.41,812.55)
(-465.76,-657.54,2762.0)	(0.12,-0.23,0.96)	(40.57,85.34,812.36)
(788.21,1090.53,2305.0)	(-0.13,0.30,0.94)	(39.49,85.91,813.72)
(-3399.98,5025.52,5731.0)	(0.28,0.46,0.84)	(37.61,86.23,816.72)



(a)

(b)

Figure 3.1: Physical configuration of (a) FaceLab eye tracker and (b) Intel RealSense stereoscopic vision system.

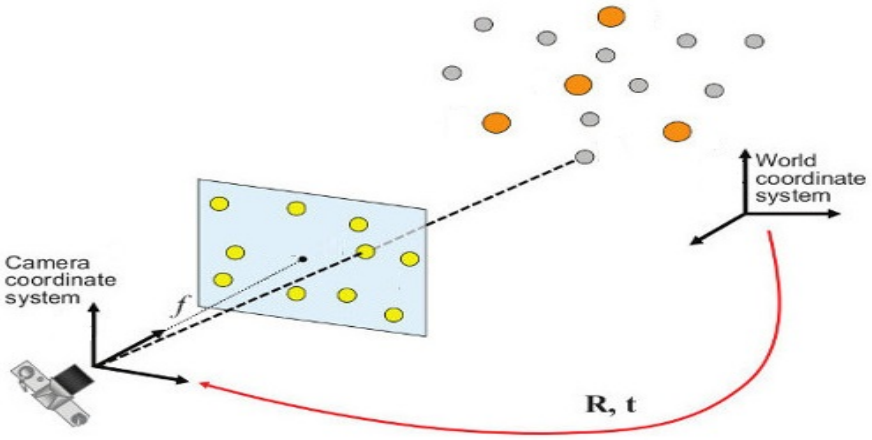


Figure 3.2: Schematic formulation of PnP problem. Obtained with modification from Ref. [37].

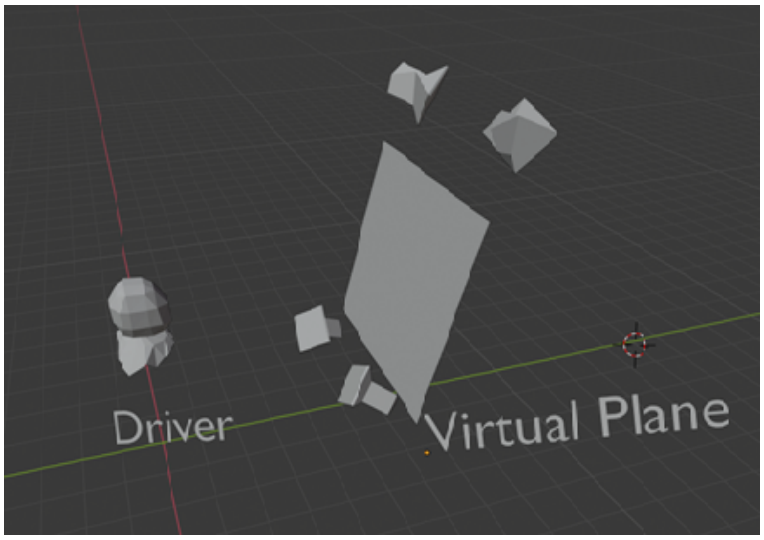


Figure 3.3: Schematic of the virtual plane configuration for the cross-calibration process between the eye tracker facing toward the driver and the stereoscopic vision system facing toward the surroundings.

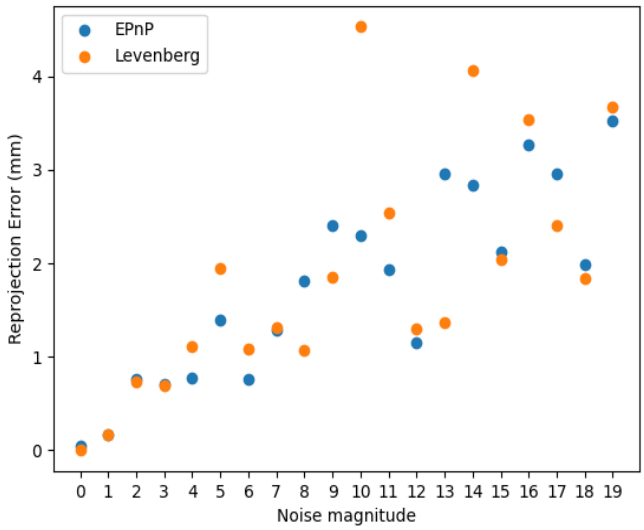
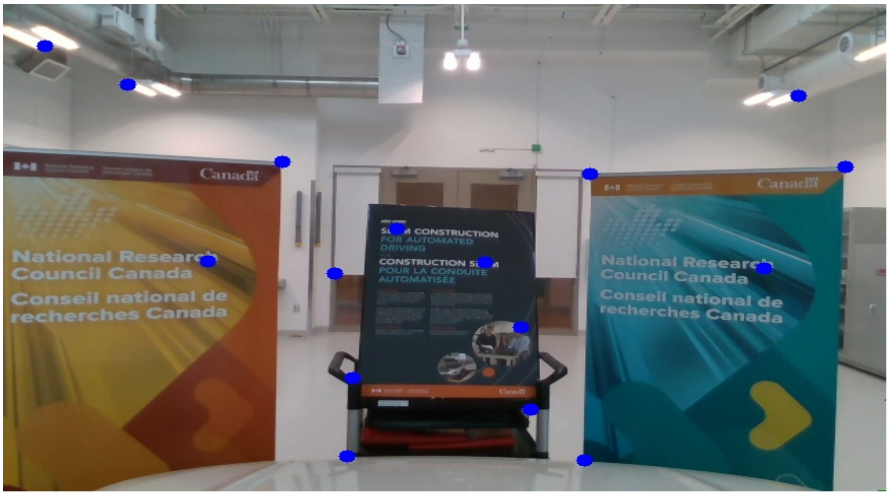
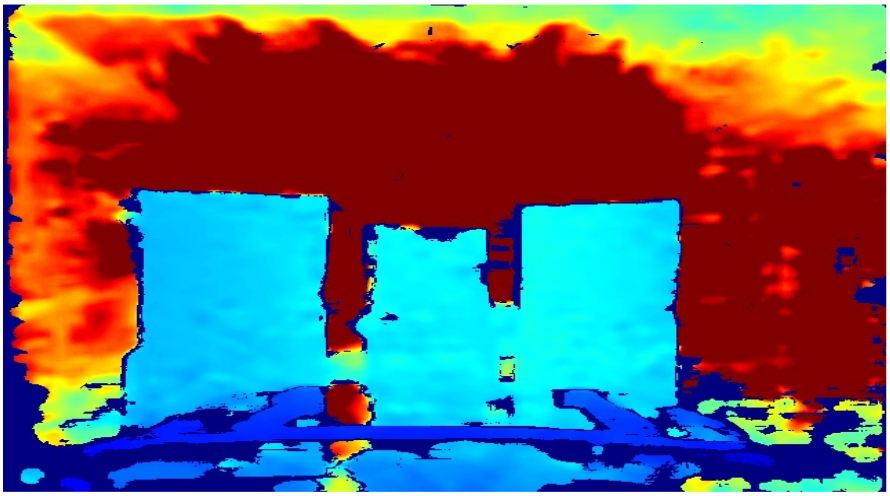


Figure 3.4: Noise robustness of PnP based on EPnP and Levenberg-Marquardt techniques applied on the synthetic data set, presented in Table 3.2.



(a)



(b)

Figure 3.5: (a) Image frame of the lab-environment place with its salient points, shown in blue points, and (b) its map depth frame.

Chapter 4

Conclusions and future work

In this thesis, we examined the PnP algorithm as a closed-form solution to the cross-calibration problem between an eye-tracker facing toward the driver's face and a stereoscopic vision system mounted on a car roof and facing toward the surrounding areas. This is a challenging problem as there is no common imaging area between the two systems. The lack of presence of such robust cross-calibration algorithms has restricted the researchers of ADASs to use either a simulator or simplify the problem by defining some areas in the vehicle that the driver is supposed to look at during a driving task.

We first applied the algorithm on a set of synthetic data set and provided a comparison between PnPRansac based on Levenberg-Marquardt and Lepetit et. al. algorithm [39], referred to as EPnP. Then, we applied the algorithm in a lab environment and showed that the reprojection error is around 100 mm and increasing the number of calibration points does not have an effect on the reprojection error. To the best of the author's knowledge, this is the first closed-form solution to the

cross-calibration problem.

There are two limitations in the present algorithm. First, the driver's head should remain constant during cross-calibration process. Violation of this assumption introduces an error whose value will be reduced as the salient points are farther away. The second limitation is that depth coverage of current vision system is around 6 to 8 meters. This means that we have two contrary effects. The reprojection would be smaller by selecting farther salient points. On the other hand, choosing farther salient points would increase the noise from the stereo vision system.

4.1 Future work

Despite past and current progress on the topics related to this thesis, there are still areas for further development and improvement on the current body of knowledge. In this regard, the following recommendations for future work are suggested:

One interesting area is cross-calibration between Lidars and stereo vision systems. In the current research, we used stereoscopic vision system to compute the point cloud of surrounding areas. However, stereo vision systems might have poor performance in low-light environments and have also poor long-distance accuracy. For these reasons, LiDAR seems a promising alternative for obtaining point clouds.

Using LiDAR would allow us to track the 3D driver gaze into the surroundings in a more comprehensive manner including at nights and in adverse weather conditions. It should be noted that the present algorithm requires pixel-wise depth estimation of the image frame, while LiDAR provides a ring point cloud of surroundings. Thus, the present algorithm requires modifications to make it suitable

for LiDAR data points.

It would be interesting to try this algorithm in real-road condition and see the robustness of the algorithm with respect to the head displacement. The present cross-calibration algorithm is based on the assumption that the driver's head remains constant and the gaze vectors are originating from the optical center of the eye tracker. However, this does not mean that the algorithm does not work in the case of driver's head movement since the effects of head displacement on the reprojection error would be smaller when the objects are farther away from the driver. It should be noted that the limitation on head displacement could be entirely removed by using a head-mounted wearable eye-tracker.

Bibliography

[1] D. W. Hansen and Q. Ji. In the eye of the beholder: A survey of models for eyes and gaze. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 3, pp. 478-500, March 2010.

[2] J. Wang, G. Zhang, and J. Shi. 2d gaze estimation based on pupil-glint vector using an artificial neural network. Applied Sciences, 6, 174, 2016, doi:10.3390/app6060174.

[3] A. Tsukada, M. Shino, M. Devyver, and T. Kanade. Illumination-free gaze estimation method for first-person vision wearable device. IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2084-2091, 2011, doi: 10.1109/ICCVW.2011.6130505.

[4] K. Takagi, H. Kawanaka, S. Bhuiyan, and K. Oguri. Estimation of a threed-imensional gaze point and the gaze target from the road images. Proc. IEEE ITSC, Washington, DC, USA, pp. 526-531, Oct. 2011.

[5] M. Kang, C. Yoo, K. Uhm, D. Lee, and S. Ko. A robust extrinsic calibration method for non-contact gaze tracking in the 3-d space. in IEEE Access, vol. 6, pp. 48840-48849, 2018.

[6] F. Lethaus, M. R.K. Baumann, F. Köster, and K. Lemmer. A comparison of selected simple supervised learning algorithms to predict driver intent based on gaze data. Neurocomputing , vol. 121, pp. 108-130, 2013.

[7] X. Li, W. Wang, and M. Roetting. Estimating driver's lane-change intent considering driving style and contextual traffic. IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 9, pp. 3258-3271, Sept. 2019.

[8] M. Mori, C. Miyajima, P. Angkititrakul, T. Hirayama, Y. Li, N. Kitaoka, and K. Takeda. Measuring driver awareness based on correlation between gaze behavior and risks of surrounding vehicles. 15th International IEEE Conference on Intelligent Transportation Systems, Anchorage, pp. 644-647 AK, 2012.

[9] T. Pech, P. Lindner, and G. Wanielik. Head tracking based glance area estimation for driver behaviour modelling during lane change execution. 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, pp. 655-660, 2014.

[10] F. Diederichs, T. Schüttke, and D. Spath. Driver intention algorithm for pedestrian protection and automated emergency braking systems. IEEE 18th International Conference on Intelligent Transportation Systems, Las Palmas, pp. 1049-1054, 2015.

[11] C. Ahlstrom, K. Kircher, and A. Kircher. A gaze-based driver distraction warning system and its effect on visual behavior. IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 2, pp. 965-973, June 2013.

[12] K. Kircher, A. Kircher, and F. Claezon. Distraction and drowsiness field operational test. VTI (Swedish Nat. Road Transport Res. Inst.), Linköping, Sweden, 2009.

[13] S. M. Zabihi, S. S. Beauchemin, E. A. M. de Medeiros, and M. A. Bauer. Frame-rate vehicle detection within the attentional visual area of drivers. IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, pp. 146-150, 2014.

[14] Statistics Canada. Canadian motor vehicle traffic collision statistics. Catalogue No. T45-3E-PDF, 2018.

[15] SAE. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. J3016,Tech. Rep., 2016.

[16] L. Petersson, L. Fletcher, N. Barnes, and A. Zelinsky. An interactive driver assistance system monitoring the scene in and out of the vehicle. IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA 2004, New Orleans, LA, USA, pp. 3475-3481 Vol.4, 2004.

[17] W. D. Montgomery, R. Mudge, E. L. Groshen, S. Helper, J. P. MacDuffie, and C. Carson. America's workforce and the self-driving future: realizing productivity gains and spurring economic growth. Securing America's Future Energy, Washington, DC, USA, Tech. Rep., 2018.

[18] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda. A survey of autonomous driving: Common practices and emerging technologies. EEE Access, 8, 58443-58469, 2020.

[19] M. Bertozzi and A. B. Gold. A parallel real-time stereo vision system for generic obstacle and lane detection. Image Processing, IEEE Transactions on, 7(1):62-81, 1998.

[20] C. Gold, M. Körber, D. Lechner, and K. Bengler. Taking over control from highly automated vehicles in complex traffic situations: the role of traffic density. Human factors, vol. 58, no. 4, pp. 642–652, 2016.

[21] N. Merat, A. H. Jamson, F. C. Lai, M. Daly, and O. M. Carsten. Transition to manual: Driver behaviour when resuming control from a highly automated vehicle. Transportation research part F: traffic psychology and behaviour, vol. 27, pp. 274–282, 2014.

[22] S. Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Tech. Rep., 2015.

[23] C. Ahlström and K. Kircher. Review of real-time visual driver distraction detection algorithms. Proceedings of the 7th International Conference on Methods and Techniques in Behavioral Research, August 2010.

[24] S. Hecker, D. Dai, and L. Van Gool. End-to-end learning of driving models with surround-view cameras and route planners. Eur. Conf. Comput. Vis. ECCV, pp. 435-453, 2018.

[25] J. Yang, C. Wang, B. Jiang, H. Song, and Q. Meng. Visual perception enabled industry intelligence: State of the art, challenges and prospects. IEEE Transactions on Industrial Informatics, 17, 2204-2219, 2021.

[26] C.H. Morimoto and M.R.M. Mimica. Eye gaze tracking techniques for interactive applications. Computer Vision and Image Understanding, 98, 1, 4-24, 2005.

[27] G J. Andersen. Focused attention in three dimensional space. Perception and Psychophysics, 47, 112-120, 1990.

[28] D. Witzner Hansen, J.P. Hansen, M. Nielsen, A.S. Johansen, and M.B. Stegmann. Eye typing using markov and active appearance models. Proc. IEEE Workshop Applications on Computer Vision, pp. 132-136, 2003.

[29] C. Hennessey and P. Lawrence. Noncontact binocular eye-gaze tracking for point-of-gaze estimation in three dimensions. IEEE Transactions on Biomedical Engineering, 56, 790-799, 2009.

[30] T. Kowsari, S.S. Beauchemin, D. Laurendeau M.A. Bauer, and N. Teasdale. Multi-depth cross-calibration of remote eye gaze trackers and stereoscopic scene systems. IEEE Transactions on Instrumentation and Measurement, Dec. 2012.

[31] F. Lethaus and J. Rataje. Do eye movements reflect driving manoeuvres? IET Intelligent Transportation Systems, vol. 1, no. 3, pp. 199–204, 2007.

[32] M. Land and J. Horwood. Which parts of the road guide steering? Nature, vol. 377, pp. 339–340, 1995.

[33] M. Land. Eye movements and the control of actions in everyday life. Prog. in Ret. and Eye Res., vol. 25, pp. 296–324, 2006.

[34] C. Gong, Z. Li, C. Lu, J. Gong, and F. Hu. A comparative study on transferable driver behavior learning methods in the lane-changing scenario. IEEE International Conference on Computer Vision Workshops (IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 2019.

[35] FaceLab5 Eye Tracker. User manual. SeeingMachine, 2009.

[36] RealSense Depth Camera D415. User manual. Intel, 2020.

[37] G. Bradski. The OpenCV Library. The OpenCV Library, 2000.

[38] M. Persson and K. Nordberg. Lambda twist: An accurate fast robust perspective three point (p3p) solver. ECCV, 2018.

[39] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o(n) solution to the pnp problem. Int J Comput Vis 81, 155, 2009.

[40] P. D. Fiore. Efficient linear solution of exterior orientation. IEEE Trans. Pattern Anal. Mach. Intell., 23,140–148, 2001.

[41] L. Quan and Z. Lan. Linear n-point camera pose determination. EEE Transactions on Pattern Analysis and Machine Intelligence, 21,774–780, 1999.

[42] A. Ansar and K. Daniilidis. Linear pose estimation from points or lines. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25,578–589, 2003.

Appendix A

Gaze data transfer over socket

This Section describes the technical steps for reading gaze data from the FaceLab Eye Tracker socket in real time, which was a challenging part of the thesis. It is hoped that this serves as a reference for future students from the RoadLab Research group.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while another socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

The first step toward this, which is transferring real time gaze via an Ethernet cable, is to run the software FaceLab 5.0, which is located on the FaceLab machine Windows 7. A.1 shows a screenshot of Windows.

After executing the FaceLab software on Windows, FaceLab cameras need to be calibrated with the following physical configuration, shown in A.2. The calibration is performed with a chessboard in different orientations, shown in A.3.

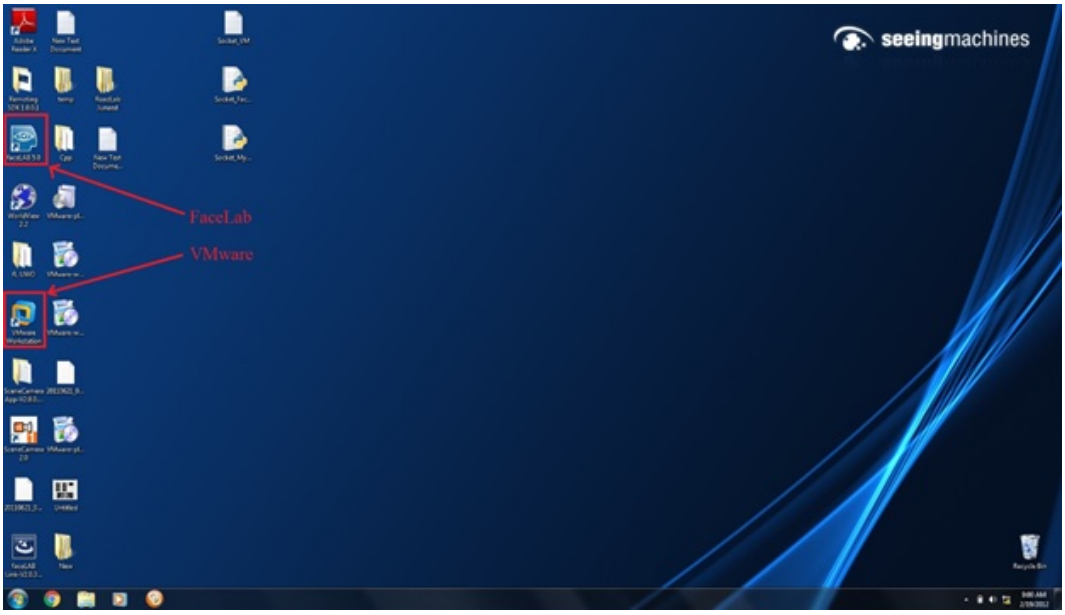


Figure A.1: Screenshot of FaceLab Eye Tracker Machine Windows, presenting FaceLab and VMware workstation icons.

After calibration, one should select the following tab to publish the gaze data on a specific IP, see A.4. IP value will be explained later in this section. There are two options for publish gaze data on a specific IP: (i) Log Accurate and (ii) Log Realtime. Log Accurate publishes data with time lag of 2.5 second, while Log Realtime publishes data with time lag of 50 millisecond. Note that the port numbers should be left as default and should not be changed.

After specifying the IP value for publishing the gaze data, one has to 'start tracking' and 'start logging' in the main menu window of FaceLab, as shown in A.4.

Afterwards, VMware machine from desktop should be executed, see Figure A.1, and select RoadLab Ubuntu to run. Find the IP of VMware and enter the IP value in FaceLab 5.0, as shown in A.4.

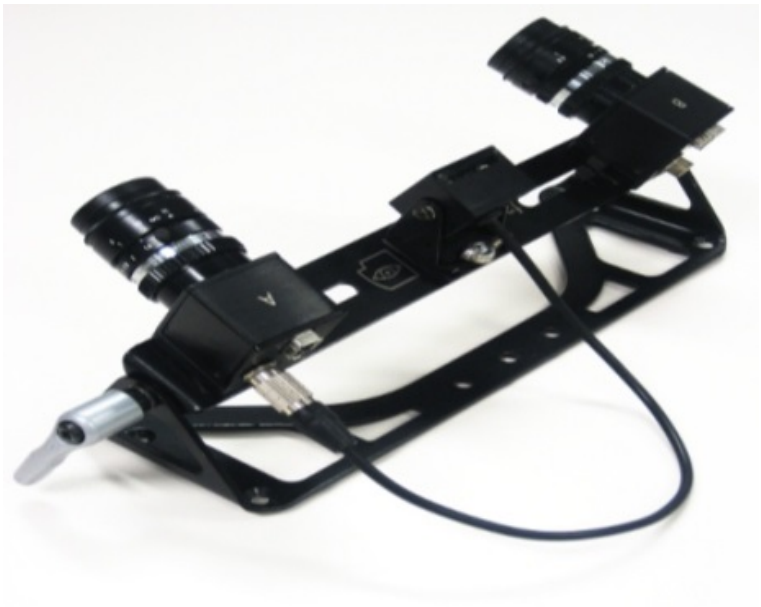


Figure A.2: Physical configuration of FaceLab cameras.

Finally, select Eclipse from the desktop (shown in A.6). Then, create a new C++ script file, inside the project *RL_Reader* that is already available. Note that if a new project is created, then many libraries need to be linked to project, which is cumbersome work. Run the following C++ code in Eclipse, reads the gaze data from the socket. After running the C++ code inside the VMware, run the following python code in the FaceLab Machine Windows 7 to receive the gaze data from virtual machine and transfer it over the socket.

// Name : Socket Programming for Transferring gaze vector, rotation vector and eye location in real time

// Author : Mohammad Karami

// Version : 1.0

// Copyright : Your copyright notice

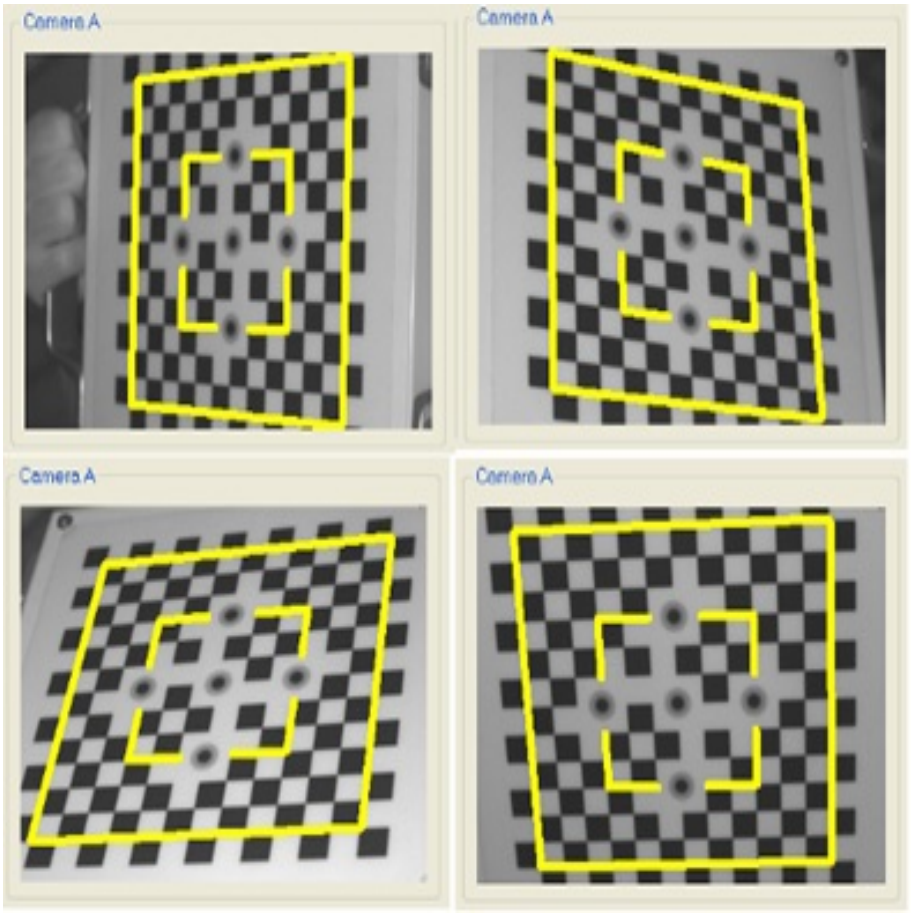


Figure A.3: Chessboard in different orientation for calibration of FaceLab cameras.

#include < *csignal* >

#include < *iostream* >

#include < *sys*/*time.h* >

#include "*RL_ReaderGUI.hpp*"

#include < *boost*/*shared_ptr.hpp* >

#include < *boost*/*algorithm*/*string.hpp* >

#include < *boost*/*chrono.hpp* >

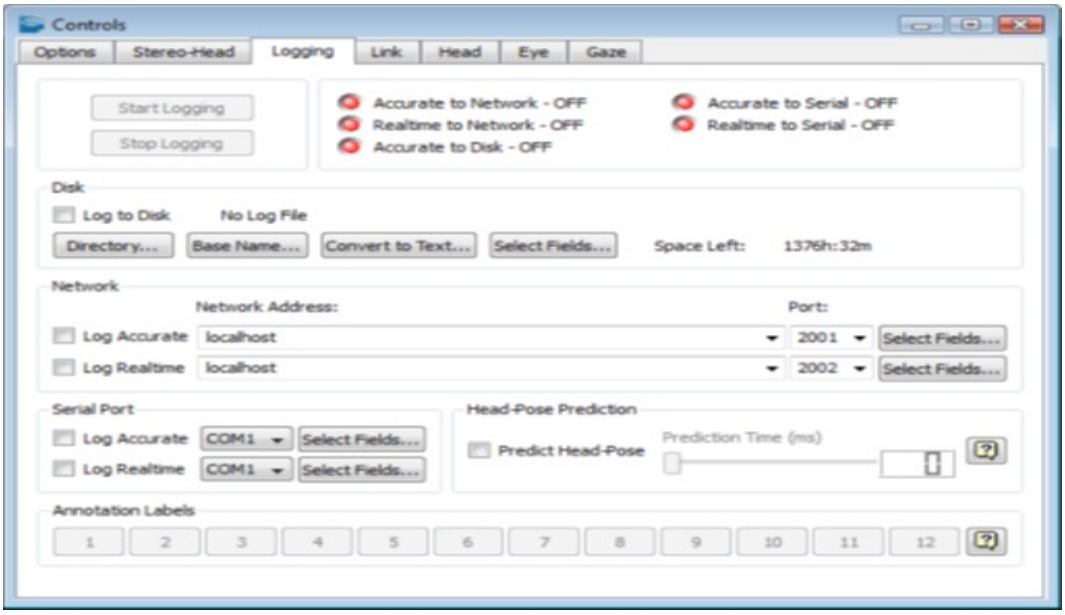


Figure A.4: Configuration of FaceLab 5.0 for publishing gaze data on a specific IP.

```
#include < sys/types.h >
#include < sys/stat.h >
#include < gtkmm.h >
#include < sys/socket.h >
#include < arpa/inet.h >
#include < unistd.h >
#include < stdio.h >
#include < stdlib.h >
#include < string.h >
#include "eod/sdk.h"
using namespace sm::eod;
using namespace sm::eod::io;
```

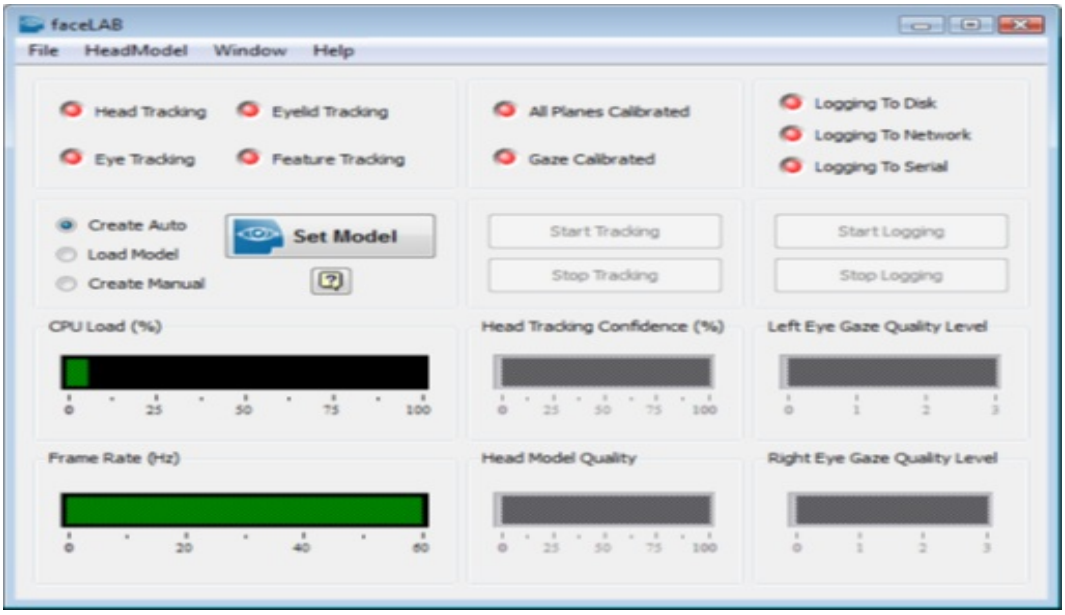


Figure A.5: Main menu window of FaceLab 5.0 for start tracking and publishing data on IP.

```
using namespace sm::eod::utils;
using namespace std;
typedef struct GazeData_sock {
    float gazeVector_x;
    float gazeVector_y;
    float gazeVector_z;
    float center_x;
    float center_y;
    float center_z;
    float rot_x;
    float rot_y;
} gaze_sock;
```

Figure A.6: Screen shot of RoadLab Ubuntu inside VMware.

```
int createSocket(int port) {
    int sock, err;
    struct sockaddr_in server;
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("ERROR: Socket creation failed");
        exit(1);
    }
    printf("Socket created");
    bzero((char *) &server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(port);
```

```
        if (bind(sock, (struct sockaddr *)&server , sizeof(server)) < 0) {
                printf(”ERROR: Bind failed”);
                exit(1);
        }
        printf(”Bind done”);
        listen(sock , 3);
        return sock;
}
void closeSocket(int sock) {
        close(sock);
        return;
}
void sendMsg(int sock, void msg, int msgsize) {
        if (write(sock, msg, msgsize) ¡ 0) {
                printf(”Can’t send message.”);
                closeSocket(sock);
                exit(1);
        }
        printf(”Message sent (%d bytes).”, msgsize);
        return;
}
int interrupt_status = 0;
void signal_handler(int param) {
        printf(” SIGINT Signal: User pressed ctr+c...”);
```

```
        interrupt_status = 1;
    }
    #define port 2002
    int main() {
        DatagramSocket inputsocket(port);
        SerializablePtr realtime_serializable;
        EngineOutputDataPtr realtimeEngine;
        InetAddress from;
        vector<uint8> buffer;
        GazeData gaze;
        int BUFFSIZE = 512;
        char buff[BUFFSIZE];
        int ssock, csock;
        int nread;
        struct sockaddr_in client;
        socklen_t clilen = sizeof(client);
        ssock = createSocket(port);
        csock = accept(ssock, (struct sockaddr) & client, &clilen);
        signal(SIGINT, signal_handler);
        float i = 0.1;
        while(interrupt_status == 0) {
            buffer.clear();
            int pos = 0;
            inputsocket.receiveDatagram(buffer,from);
```

```
if(buffer.size() > 0)
cout << ”Data Received from buffer” << buffer.size() << endl;
} else
cout << ”Not received” << endl;
if(buffer.size() > 0) {
realtime_serializable = SerializableFactory::newObject(buffer, pos);
realtimeEngine = boost::dynamic_pointer_cast<EngineOutputData>(realtime_serializable);
}
EngineOutputData  engine = realtimeEngine.get();
EngineOutputData &facelabData = engine;
EyeId eyeId = RIGHT_EYE;
GazeOutputData & faceGaze =  facelabData.eyeOutputData()− >gazeOutputData();
const fStdVector2 & rot = faceGaze.gazeRotation(eyeId);
const fStdVector3 & center = faceGaze.eyeballCentre(eyeId);
gaze.quality = faceGaze.gazeQualityLevel(eyeId);
printf(”%f %f” , rot[0], rot[1]);
float c1 = cos(rot[0]);
float s1 = sin(rot[0]);
float c2 = cos(rot[1]);
float s2 = sin(rot[1]);
gaze.gazeVector[0] = s1  c2;
gaze.gazeVector[1] = -s2;
gaze.gazeVector[2] = c1  c2;
gaze.eyeCenter[0] = 1000  center[0];//inmillimeter
```

```
gaze.eyeCenter[1] = 1000*center[1];
gaze.eyeCenter[2] = 1000*center[2];
printf("%1.5f %1.5f %1.5f", gaze.gazeVector[0], gaze.gazeVector[1],
gaze.gazeVector[2]);
printf("%1.5f %1.5f %1.5f", gaze.eyeCenter[0], gaze.eyeCenter[1],
gaze.eyeCenter[2]);
//Transferthefilesonthesocket
gaze_sock p = new gaze_sock;
i += 0.1;
p- >gazeVector_x = gaze.gazeVector[0];
p- >gazeVector_y = gaze.gazeVector[1] ;
p- >gazeVector_z = gaze.gazeVector[2];
p- >center_x = gaze.eyeCenter[0];
p- >center_y = gaze.eyeCenter[1];
p- >center_z = gaze.eyeCenter[2];
p- >rot_x = rot[0];
p- >rot_y = rot[1];
sendMsg(csock, p, sizeof(gaze_sock));
delete p;
} //endofwhileloop

printf("Closing Socket");
closeSocket(ssock);
closeSocket(csock);
```

```
    return 0;
}
```

Python code for receiving and transferring gaze data over socket (should be executed inside Windows 7 of FaceLab Mahcine:

```
import socket
import sys
from ctypes import
import time
HOST = '192.168.127.129' # The server's hostname or IP address, as sample
PORT = 2002 # The port used by the server
class GazeData(Structure):        _fields_ = [("gazeVector_x", c_float),
    ("gazeVector_y", c_float),
    ("gazeVector_z", c_float),
    ("center_x", c_float),
    ("center_y", c_float),
    ("center_z", c_float),
    ("rot_x", c_float),
    ("rot_y", c_float)]
server_addr = (HOST, PORT)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(server_addr)
print("Connected to VMware")
print("Waiting for another PC to connect")
new_s = socket.socket()
```

```
new_s.bind((”, PORT))
new_s.listen(5)
c, addr = new_s.accept()
try:
    while True:
        buff = s.recv(sizeof(GazeData))
        GazeData_in = GazeData.from_buffer_copy(buff)
        print (”Received x=%f, y=%f, z=%f” % (GazeData_in.gazeVector_x,
        GazeData_in.gazeVector_y, GazeData_in.gazeVector_z,))
        c.sendall(GazeData_in)
        #time.sleep(0.01)
except KeyboardInterrupt:
    print(”Closing Socket”)
    c.close()
    s.close()
    new_s.close()
```

Now, you can use the following Python code in any machine that is connected with Ethernet cable to the FaceLab Machine to read the gaze vector, rotation vector and eye location in real time:

```
import socket
from ctypes import
port = 2002
HOST = ’192.168.65.127’
class GazeData(Structure):
```

```
        _fields_ = [("gazeVector_x", c_float),
        ("gazeVector_y", c_float),
        ("gazeVector_z", c_float),
        ("center_x", c_float),
        ("center_y", c_float),
        ("center_z", c_float),
        ("rot_x", c_float),
        ("rot_y", c_float)]
s = socket.socket()
s.connect((HOST, port))
print("Connected to FaceLab Machine")
try:
        while True:
                buff = s.recv(sizeof(GazeData))
                GazeData_in = GazeData.from_buffer_copy(buff)
                print ("Received x = %f, y = %f, z = %f" %
                (GazeData_in.gazeVector_x,
                GazeData_in.gazeVector_y,
                GazeData_in.gazeVector_z))
                //GazeData_in.center_x,
                //GazeData_in.center_y,
                //GazeData_in.center_z,
                //GazeData_in.rot_x,
                //GazeData_in.rot_y))
```

```
except:
        s.close()
        print("Socket closed")
```

Curriculum Vitae

Name:	Mohammad Karami
Post-Secondary Education and Degrees:	University of Western Ontario Computer Science Department 2019-2021 M.Sc. University of Western Ontario Civil and Environmental Engineering Department 2015 - 2019 Ph.D. Isfahan University of Technology, IRAN Mechanical Engineering Department 2010 - 2013 M.Sc. Persian Gulf University, IRAN Mechanical Engineering Department 2005 - 2010 B.Sc.
Honours and Awards:	Graduate Research Assistant 2015-2021
Related Work Experience:	Teaching Assistant The University of Western Ontario 2016 - 2021 Machine Learning Engineer National Research Council 2020-2020

Publications:

- M. Karami, H. Hangan, L. Carassale, H. Peerhossaini, Coherent structures in tornado-like vortices, Selected paper by Editor, Physics of Fluids 31, 085118 (2019).

- M. Karami, H. Hangan, L. Carassale, Statistical and modal analysis of surface pressure fluctuations in tornado-like vortices, Selected paper by Editor, Physics of Fluids 32, 075109 (2020).