Technische Universität Berlin

Institut für Telekommunikationssysteme
Fachgebiet Verteilte offene Systeme

Fakultät IV
Ernst-Reuter-Platz 7
10587 Berlin

Masterarbeit

Tackling Communication Delays for Tele-Operated Driving

Kosta Ivancevic

Berlin, March 31, 2020

Betreuer:
Prof. Dr. Manfred Hauswirth

Matrikelnr: 397172

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Berlin, March 31, 2020

_____

(Signature)

Deutscher Akademischer Austauschdienst
Kennedyallee 50
D-53175 Bonn

FOKUS Institute
Kaiserin-Augusta-Allee 31
10589 Berlin

First of all, I would like to express my gratitude to my supervisor, Kay Massow from the Fraunhofer FOKUS Institute for allowing me to carry out state of the art research in the field of Tele-Operated driving. I have been particularly lucky to have a supervisor who had such a positive influence on improving my problem-solving skills, who responded to all my problems with proper guidance, and without whom this thesis would not have been so comprehensive. My supervisor, Kay Massow, has made this thesis a delightful journey.

Furthermore, I would use this opportunity to thank Oliver Sawade, who has been my supervisor during another project with Daimler Center for Automotive IT Innovations and without whom I would not have the opportunity to apply Machine Learning to the automotive applications.

This thesis was originated in cooperation with the Fraunhofer Institute for Open Communication Systems (FOKUS). I intend to extend my deepest gratitude to the Fraunhofer FOKUS for providing me with the opportunity to participate in this project.

Furthermore, throughout my studies, I was generously supported by the Deutscher Akademischer Austauschdienst (DAAD). This scholarship programme has not only enabled me to come to the Technische Universität Berlin, but also to experience the German language and culture and to have an international experience for a lifetime to remember.

Abstract

Fundamentally, Tele-Operated driving corresponds to operating a vehicle from a remote location. The vehicle uses cameras and other sensors to provide a remote driver with the necessary information about the vehicle surroundings. This information is sent via the network and presented to the driver. By relying on the information sent from the vehicle, the driver is responsible for generating a set of commands. The server then sends these commands via the network to the vehicle. Upon receiving the commands, the vehicle applies these commands, thus completing the driving cycle. The iterative behaviour of the system reflects itself in a sequence of sensory information sent from the vehicle to the server, and a sequence of commands received.

The vital part of the system is the network that connects the vehicle and the server. Unfortunately, the negative property of any network is that it introduces latencies. Generally, two entities that communicate in different locations are unable to transmit messages with no latency. In other words, the information that is sent by one entity is received by another entity with an inevitable delay. Analogously, the remote driver is presented with outdated information because the new information is still traversing the network. Similarly, the commands that are applied by the vehicle differ from the commands applied by the driver. The problem that is tackled by this thesis is minimizing the negative influences of network delays.

One approach to reducing the effects of network delays is to use predictions to estimate the probable trajectories of the surrounding vehicles, thus compensating the delays. The server-side of the system implements this approach in the form of various prediction mechanisms, that range from deterministic velocity-based predictions to machine-learning-based approaches like regression or neural networks. Positions of the traffic participants can be accurately computed by using the prediction mechanisms as mentioned above. These predictions span into the future, which corresponds to the latencies, that arise in the networks. Practically, the predictions enable the remote driver to observe the objects in positions that resemble future behaviour, thus effectively eliminating the network delays.

Another way to ensure that the adverse effects of outdated information are minimal is to use prediction mechanisms on the vehicle side to estimate the unreceived commands. The approach based on Artificial Intelligence may be used for this purpose. We have designed, trained and evaluated neural networks to use a sequence of driving commands to generate a set of commands that resemble the commands that are currently applied by the remote driver at the server-side. In other words, we have designed a prediction mechanism that effectively estimates the currently applied commands, thus outperforming traditional Tele-Operated driving systems.

Zusammenfassung

Tele-Operated Fahren entspricht im Wesentlichen der Bedienung eines Fahrzeugs von einem entfernten Standort aus. Das Fahrzeug verwendet Kameras und andere Sensoren, um einem Fernfahrer die notwendigen Informationen über die Fahrzeugumgebung zur Verfügung zu stellen. Diese Informationen werden über das Netzwerk gesendet und dem Fahrer präsentiert. Indem er sich auf die vom Fahrzeug gesendeten Informationen verlässt, ist der Fahrer für die Generierung eines Befehlssatzes verantwortlich. Der Server sendet diese Befehle dann über das Netzwerk an das Fahrzeug. Nach dem Empfangen der Befehle wendet das Fahrzeug diese Befehle an und beendet damit den Fahrzyklus. Das iterative Verhalten des Systems spiegelt sich in einer Folge von sensorischen Informationen wider, die vom Fahrzeug an den Server gesendet werden, und einer Folge von empfangenen Befehlen.

Der entscheidende Teil des Systems ist das Netzwerk, das das Fahrzeug und den Server verbindet. Leider ist die negative Eigenschaft eines jeden Netzwerks, dass es Latenzen einführt. Im Allgemeinen können zwei Einheiten, die an verschiedenen Standorten kommunizieren, keine Nachrichten ohne Latenzzeit übertragen. Mit anderen Worten, die Informationen, die von einer Einheit gesendet werden, werden von einer anderen Einheit mit einer unvermeidlichen Verzögerung empfangen. Analog dazu wird dem entfernten Fahrer eine veraltete Information angezeigt, da die neue Information immer noch das Netzwerk durchläuft. Ebenso unterscheiden sich die vom Fahrzeug ausgeführten Befehle von den vom Fahrer ausgeführten Befehlen. Das Problem, das durch diese Arbeit angegangen wird, ist die Minimierung der negativen Einflüsse von Netzwerkverzögerungen.

Ein Ansatz zur Reduzierung der Auswirkungen von Netzverzögerungen besteht darin, anhand von Vorhersagen die wahrscheinlichen Trajektorien der umliegenden Fahrzeuge zu schätzen und so die Verzögerungen auszugleichen. Die Serverseite des Systems implementiert diesen Ansatz in Form verschiedener Vorhersagemechanismen, die von deterministischen, geschwindigkeitsbasierten Vorhersagen bis hin zu maschinenlernbezogenen Ansätzen wie Regression oder neuronalen Netzwerken reichen. Die Positionen der Verkehrsteilnehmer können mit Hilfe der oben genannten Vorhersagemechanismen genau berechnet werden. Diese Vorhersagen reichen in die Zukunft, was den Latenzen entspricht, die in den Netzwerken entstehen. In der Praxis ermöglichen die Vorhersagen dem entfernten Fahrer, die Objekte an Positionen zu beobachten, die dem zukünftigen Verhalten ähneln, wodurch die Netzwerk- Verzögerungen effektiv beseitigt werden.

Eine weitere Möglichkeit, sicherzustellen, dass die negativen Auswirkungen veralteter Informationen minimal sind, besteht darin, Prognosemechanismen auf der Fahrzeugseite zu verwenden, um die nicht erhaltenen Befehle zu schätzen. Zu diesem Zweck kann der auf künstlicher Intelligenz basierende Ansatz verwendet werden. Wir haben neuronale Netzwerke entwickelt, trainiert und evaluiert, um eine Folge von Fahrbefehlen zu verwenden, um einen Befehlssatz zu erzeugen, der den Befehlen ähnelt, die derzeit vom Remote-Treiber auf der Serverseite angewendet werden. Mit anderen Worten, wir haben

einen Vorhersagemechanismus entwickelt, der die aktuell verwendeten Befehle effektiv schätzt und damit die traditionellen teleoperativen Antriebssysteme übertrifft.

Contents

List of Figures

List of Tables

1 Introduction

Although the overall end-goal for road traffic in the world is a fully autonomous vehicle, a comprehensive commercially available solution that fulfils all the requirements regarding safety, usability and scalability are not yet present. Failure to meet all the requirements is partly present because the produced systems have not yet reached the desired performance for full autonomy in all circumstances, as well as the limiting regulatory policies, which are still relatively underdeveloped [4]. A step towards fully autonomous driving may be seen in the concept of Tele-Operated driving, where the vehicle is operated from a remote location by a human operator. The driver in the cockpit is not directly in charge of applying commands. In general, they should not be able to tell the difference between Tele-Operated and autonomous driving. Likewise, from the passenger standpoint, both approaches that exclude direct operation should be at least as good as human driving for any metric.

Although vehicle Tele-Operation was defined in [5] as simply "operating a vehicle at a distance", we adopt a more specific definition.

> *Vehicle Tele-Operation is a concept of indirectly applying commands from a remote location without direct audio-visual information. The operator is provided with sensory information from the vehicle, such as image camera stream and audio signal using communication networks to transfer this information to the remote location. The vehicle receives the commands, which are applied accordingly, as a stream of parameters through communication channels.*

Figure 1.1 depicts two conceptual diagrams, corresponding to the Tele-Operation systems. Block diagram 1.1a describes the fundamentals of Tele-Operation.

We observe that it consists of three entities

- Vehicle - That is responsible for collecting vehicle sensory information[1], transmitting it to the server and, upon reception, applying received commands.
- Server - That is used to provide the remote driver with according information and generate the commands from a distant location based on the vehicle information that is received.
- Network - That enables client and server to communicate. More precisely, the network component is used as a medium for the vehicle to share its sensory infor-

[1] In our work; we handle image streams and positional information as vehicle data. Without loss of generality, this can be combined with audio signals, LiDAR point cloud, etc

(a) The block diagram of a Tele-Operation system consisting of a vehicle that is operated via network by the teleoperator.

(b) The block diagram of a Tele-Operation system proposed in [6]. The vehicle uses mobile networks to transmit information to the controller cocpkit, from where the commands are transmitted back.

Figure 1.1: Conceptual schemes of Tele-Operation systems

mation with the server and to receive the generated commands, corresponding to the vehicle information.

Diagram 1.1b describes the Tele-Operation system similar to ours, introduced in [6]. We observe that they are conceptually the same as they implement the same task and encounter the same problems regarding network delays which cause the vehicle information.

The concept of Tele-Operation as a means of controlling vehicles dates back to the 1900s, although the increase in its usability was not significant before 1970's.[5] The popularity of Tele-Operation vehicles rose ever since and is more and more incorporated into our everyday lives. Although we will focus only on ground vehicles, systems are not only limited to these but also aircraft, boats and other vehicle types that use the ground, water or air as a medium of transport.

1.1 Motivation and Use Cases

Tele-Operated driving is an important concept and developments in this area are mainly focused on increasing practicality and ensuring the safeness of autonomous driving. Entirely autonomous vehicles are not yet in a development phase where the driving process is fully resistant to errors and state of the art systems are still underperforming when compared to human driving. Therefore, it is necessary to keep the human in the loop at least until the system can outperform humans. Tele-Operated driving imposes itself as a "transition" solution during this period.

Although the use cases for Tele-Operated driving are diverse, we point out some of them to stress the importance of automobile development in this direction. Tele-Operated driving can be used as a:

- Failsafe mechanism. In certain hazardous situations, where various malfunctions may occur, it is necessary for a vehicle to enter a safe state. For example, this may imply parking a vehicle on the side of the road when autopilot exhibits unwanted behaviour.

- Trajectory correction mechanism. In some cases, the automatic trajectory estimation, computed by the autonomous autopilot, may be impracticable. In such cases, human correction is necessary in the form of manual trajectory correction.

- Opportunity to overcome vehicle-human interaction. Since humans are still the major traffic participants and the traffic rules are created by humans. Therefore a human Tele-Operater is necessary to be in the loop.

- "Vehicle on Demand" concept. An interesting concept for Tele-Operated driving may arise in cases when a potential user requires a rent-a-car for a certain amount of time or if multiple users participate in car sharing. In both cases, as well as several further ones, Tele-Operated driving may be used to drive the vehicle from one user to another.

- Vehicle valet parking. Instead of searching for a parking spot, the user may be presented with an opportunity to leave the parking task to the remote operator. Tele-Operated Valet parking may be a foundation for various parking space optimizations.

- Edge-case rule braking mechanism. In certain cases, the autonomous autopilot may avoid rule-breaking, although it may be necessary. In order to overcome the "fear of penalizing"[2], the human is kept in the loop.

Tele-Operated driving can aid autonomous driving for performing tasks for which vehicle cannot perform at all or with sufficient safety requirements. For example, in the case of vehicle malfunction, human coordination is a necessary failsafe mechanism. Furthermore, an engaging use case for Tele-Operated driving can be seen when performing actions, for which the risk of autonomously performing the task is too high. Humans may be better at assessing risk than machines. Depending on the situation, the operator may only be needed for confirmation of the desired action or manual control is required.

Problems regarding trajectory are the main reason for Tele-Operated driving. Dirt or gravel roads are often harder to distinguish and may not be shown on maps. Furthermore, the available map information may be outdated, and therefore, the vehicle may not be able to locate itself on the map. Likewise, there may be a need to use unconventional means of transport, such as transport boats. In such situations, manual vehicle control surpasses autonomous driving. The trajectory of a vehicle can be blocked by an object that can't be properly classified. Furthermore, it may be another vehicle due to malfunction blocking the road. In some cases, the vehicle is unable to recalculate the new trajectory. Situations like the former provide two solutions:

- Park the vehicle safely on the side of the road and wait until the desired trajectory becomes available. Unfortunately, in cases in which the map information is

[2]By the term "fear of penalizing", we consider the state in which autopilot refrains from rule-breaking actions although they may be necessary in some edge cases. For example, if the road, marked by the continuous double line is blocked by an immovable object

outdated, this is not a valid option. Therefore, in order to avoid the inability to enter the safe state, the command is transferred to the human driver as a failsafe mechanism.

- Transfer the command to the Tele-Operator, who ether specifies the new desired trajectory or takes continuous control of the vehicle

A significant part of vehicle decision making comes from utilizing camera image feed with Machine Learning techniques like DNNs, which is due to the recent progress in this area of science. Unfortunately, these systems are not yet trajectory estimators immune to errors. A concrete example can be seen on figure 1.2. Tian et al. presented a paper [1] in which they found that small variations and perturbations in the camera images produce trajectory errors. Although they propose a verification and evaluation mechanism, problems which they encountered may pose Tele-Operation as a useful concept in certain scenarios, where the certainty, that trajectory is correct, is below some threshold.

Figure 1.2: An example of an errournous behaviour found in [1] when insignificant modifications are applied to camera feed. Errors as such pose a good reason for Tele-Operation driving where, an operator directs a correct trajectory.

Since humans create traffic, it can be prone to human errors in terms of contradicting or inconsistent traffic rules. Even confusing sensory inputs make it difficult for a machine to perform autonomously, and therefore human assistance is needed. Further, some scenarios can be considered "corner cases" because traditional traffic rules are not directly applicable. For example, a traffic light might be out of order, and the policeman can replace its functionality. In such scenarios, the Tele-Operator may be more familiar with the required actions, than the trajectory planner in the vehicle itself. Furthermore, there are cases where a crash scene is handled by multiple policemen, where they simultaneously direct the traffic. A human operator might find it easier to find and maintain the focus on the primary policeman.

Similarly, driving through a crowd of people can be challenging for a vehicle, but straightforward for a human operator. Likewise, some scenarios such as *"reißverschlussverfahren"* and moving vehicles for the ambulance or police car to successfully pass can be better handled by people.

Figure 1.3: An image from [2] of a policeman directing traffic.

On the other hand, Tele-Operation broadens the possibilities of car sharing. Let us consider the following example. A person uses the car to go between points A and B, where the journey ends. Afterwards, another customer requests a vehicle to drive between points C and D. Tele-Operated car sharing would imply that a Tele-Operator it takes control of the vehicle to transport it from point B to point C. Traditional car sharing, such as DriveNow and Car2go, would benefit from Tele-Operation because it would reduce the overall number of needed vehicles to cover user demand. Similarly, the concept of Tele-Operation would enable distributing vehicles to locations with the highest demand.

Likewise, for people that perform jobs with fixed and standard working times, during which they do not require a vehicle, another person may be able to use the vehicle. Similarly, transitions between customers can be operated remotely. The benefits of car sharing include both reducing the number of vehicles, as well as freeing the parking spaces, which are required to a greater extent when this concept is not used.

In contrast to the applications where Tele-Operated driving is used as a driving aid. There are scenarios where it is directly applicable, most notably, Valet parking. People spend a significant amount of time going to the parking lot to begin their journey, as well as searching for parking at the destination. Tele-Operation could be used dually to solve this problem. Before the ride commences, Tele-Operator is responsible for driving a car from some remote parking location to the user. As a result, the parking space distribution can be theoretically optimized by fetching a vehicle from a location where the demand for parking spaces is high. Upon finishing a journey, the user leaves the vehicle, thus handing over the control to the Tele-Operator. Afterwards, Tele-Operator parks the vehicle on a

suitable spot. Similar parking optimizations can be achieved because the operator may choose parking spaces with lower demand, thus reducing congestion. Overall the Tele-Operated valet parking concept would save a significant amount of time and pave the way towards a collaborative paring in which vehicles communicate mutually to reach the most optimal parking arrangement and utilization. The development of WSNs has led to the possibility of equipping parking lots with small and relatively low-cost modules that monitor the availability of parking places. A network, formed by a set of interconnected nodes, may be used to form a virtual parking map of the broad area. Systems proposed in [7] and [8] introduce similar ideas and a comprehensive comparison study [9] analyzes the concept in greater detail. Motivated by these advancements, we argue that Tele-Operator, being equipped with a virtual parking map may be able to reduce the overall congestion and to have a contribution in the evening the demand between some traffic areas by parking in areas with lower demand.

Finally, some scenarios are considered "corner cases" because they occur in only in extraordinary situations. Due to the unexpected nature of these events, autonomous vehicles might not be fully capable of safely fulfilling the task. Often, to safely overcome some given scenario, the vehicle may be required to circumvent some rules, which would, in typical cases, be strictly penalized. An example of these exceptions would be a situation, in which an immovable object blocks a lane. The vehicle may be forced to change to the opposite lane over a continuous double line. Human interaction may be needed to asses the possibilities and constraints and to conduct this manoeuvre. Analogously, avoiding sinkholes, which often occur in some parts of the world may require human interaction to secure passenger safety.

As various use cases, some of which are stated above indicate, Tele-Operation is a concept with multiple applications. Unfortunately, implementing real-world, reliable, fail-proof systems that achieve a high level of redundancy is a challenging task. The foundation of most issues is the presence of latency in the networks. For example, transmitting video feed between the vehicle and the operator has to be done in real-time, with as low latency as possible.[10] Furthermore, although the progress in this area is clearly visible, wireless communication has not yet achieved the desired level of reliability. [11] [12] For the Tele-Operation applications, the signal strength has to be consistently above a certain threshold. Finally, the area coverage must not contain fields that are poorly covered or not covered at all.

1.2 Objective

Motivated by the previous chapter, we are convinced that Tele-Operation concept is a problem worth tackling. Although an optimal and fully fault-free solution may not yet be reached, this master thesis aims to find an improvement point and propose a feasible solution that would yield increased performance. Let us we focus again on Figure 1.1a. It is observable that the main optimization points may be in the following aspects: vehicle, server, and network for communication. Further, we can observe the following:

The network delays occur in every network, and they represent the time

needed for information sent from its source to reach its designated destination. In terms of Tele-Operated driving, network delays cause the vehicle information, received by the server, to be outdated. In other words, current locations of the vehicle and all surrounding traffic participants differ from the information that is available to the server as a result of network delays. Likewise, the commands that are applied to the vehicle-side differ from the commands applied to the server-side because the current commands require a certain amount of time to reach the designated vehicle.

In this work, we simultaneously tackle both the negative consequences of network delays. We propose the following two approaches.

Positional Prediction Approach is an approach in which we aim to tackle delay-caused issues at the server-side. More precisely, we design a mechanism for predicting future positions of all traffic participants. This conceptual system allows for a user, operating a vehicle at a distance, to compensate for the network delays by predicting the future positions of all vehicles in a scene, thus generating a scene that is similar to the scene currently observed at a vehicle-side. We implement this approach using different methods, which are, linear and polynomial regression, deterministic motion vector prediction and machine learning approach based on neural networks. We evaluate the correctness of each method by measuring the difference between the actual scene and the prediction. Furthermore, we investigate the influence of the noise on all methods.

Command Predicting Approach is a concept in which we aim to make predictions on a vehicle-side, thus tackling the delay-related issues that produce command disparity. More precisely, we rely on artificial neural networks as machine learning concepts to predict commands that are not yet received, based on a sequence of commands that are received by the vehicle up to the point of prediction. We compare the case without using predictions with the case where we use artificial neural networks to generate predictions.

In order to assure and measure the quality of our proposals, evaluation is one of the crucial parts of this thesis. As described in section 3.3, evaluation is specific and custom-designed for each Tele-Operated proposal, and therefore available in sections 4.4 and 4.4.

During our research, we identify the following features that a promising positional prediction system should fulfil.

- **Accuracy** The prediction mechanism is expected to generate predictions that fully or, to some large extent, correspond to the set of positions that a vehicle finds itself in through time. In other words, the predicted trajectory should match the real trajectory of a vehicle to the largest possible extent.

- **Usability** The system should be usable from the standpoint of a user. The user should feel the ride as "smooth" and continuous.

- **Scalability** The computational intensity required for generating adequate predictions should scale well with the increased traffic.

Similarly to the position prediction approach, when generating the commands on the vehicle-side of the Tele-Operated system, we have identified the **accuracy** as the key optimizing constraint. In other words, the commands that are predicted should match the ones currently applied by the human operator to the greatest extent possible. Therefore we organize the evaluation of our work in this direction; to compare the difference between human- and machine-generated commands.

1.3 Scope

By observing components of block diagrams on Figure 1.1, three components impose themselves as core elements of the Tele-Operation system. Therefore, client, server and a network in between are the main optimization points which may have a contribution towards increasing the overall performance. Likewise, the challenges of implementing such a system may arise in all three subsections. Any vehicle malfunction has a direct impact on driving performance. Given that vehicle can underperform or fail regardless of the complete Tele-Operation implementation, such challenges and problems are not further discussed. Likewise, the network architecture, interconnecting client and server, is assumed to produce unexpected delays that we take for granted and provide no conceptual improvement for the network architecture. The contribution of this work is to demonstrate, that network latency problem, that causes outdated information in the server-side and outdated commands on the vehicle-side may be compensated using machine learning.

For the initial steps, we design and implement the following components in PHABMACS simulator that can be seen as prerequisites for any Tele-Operation concept.

- vehicle that is implemented to inform the server about the current scene information and apply commands that are received from the server. Gathering, encapsulating and transmitting inadequate information form are subroutines that comprise information task. Receiving, maintaining and applying commands received from the server comprise a driving task.
- server is in charge of receiving the information from the vehicle, presenting it to the driver and returning a set of messages with commands.
- Message system is a set of stack components that maintain a sequence of most recent messages for both server and client. Furthermore, specific message design is provided for each of the two approaches.
- Delay and noise generators are components that are added as a source of error in our systems. We evaluate how our proposals are immune to both components. Both components are implemented as several generators that produce numbers according to the Gaussian function, given mean and variance. We interpret delay generator as selecting a random delay around some mean value, while we add the random noise to all the measurements in the system to examine the system immunity to the measurement noise.

On top of these components, we implement both approaches.

As introduced in section 1.2, positional prediction approach aims at using deterministic and machine-learning-based predictions to predict the positions of all traffic participants that surround the vehicle, including the vehicle itself. We design a conceptual system in which the client vehicle collects information from the surrounding objects[3]. The information, gathered by the vehicle is then encapsulated and sent to the server components. On the server-side, this information is received with a random delay, corresponding to network latency. To estimate the actual positions, we use the received information to generate position predictions for all objects that correspond to the time, at which the commands are expected to reach the vehicle. For this purpose, we develop this approach using multiple implementations:

- **Linear and Polynomial Regression** Using regression methods, we aim to extrapolate the motion of the vehicle by fitting a straight line or a polynomial of some degree D to the set of N most recent measurements. The parameters of the model are fitted to minimize the squared error between the model and the received motion parameters. The advantage of this technique, in comparison to the simple baseline model, is the ability to capture complex motion behaviour, that vehicle may exhibit, while the baseline model is characterized by simple predictions using only single most recent positional change.

- **Recurrent Neural Network** This method uses RNNs[3], more precisely, LSTMs. From the sequence of the N most recent velocity and rotation rate measurements, the future motion is predicted using this DNN approach.

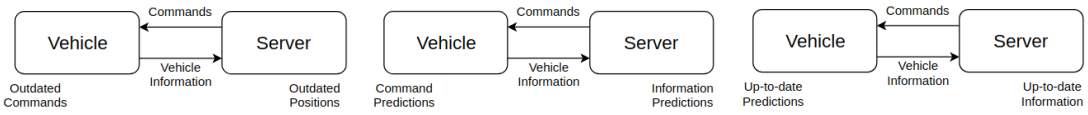
- **Deterministic Motion Vector** This variant of the positional prediction uses current angular and lateral velocity and acceleration joint with position to estimate the future trajectory of the vehicle.

Finally, we design a mechanism that enables user input for the commands, implemented as a game controller with custom added mini wheel.[4]

As introduced in section 1.2, command prediction approach aims at using machine-learning-based predictions to predict the commands that are not yet received by the vehicle due to network latency. Given that the main characteristic of a command sequence is that there exists a time-dependancy, we implement our approach using LSTM version of recurrent neural networks. Upon creating, training and evaluating our neural networks externally, we import them to PHABMACS simulator as graphs and use them to generate predictions. The success of using this approach is measured by comparing how similar are commands between the server and client when predictions are used, as opposed to the case when no predictions are used. In an ideal case, the commands on both sides of the network interface should be the same, which would imply that the network latency does not influence the Tele-Operation.

[3] In our implementation, the objects are only vehicles. Without the loss of generality, the scope of objects may be extended with objects as pedestrians, bicycles, etc

[4] For a detailed description see section 3.1.



(a) Conceptual Tele-Operation block diagram when predictions are not used

(b) Conceptual Tele-Operation block diagram when predictions are used

(c) Conceptual Tele-Operation block diagram when network delays are negligible

Figure 1.4: Overview of different conceptual scenarios with respect to the presence of network delays and their influence

The core contribution of this thesis is a proof of concept system that would demonstrate how to tackle latency related issues. Three distinct scenarios are illustrated in Figure 1.4. The Figure 1.4a corresponds to the scenario where no predictions are used. The disproportion between the scene, available on the server-side and the current scene on the vehicle-side is directly proportional to the network latency in upward[5] direction. Likewise, the commands applied on the server-side are applied on the vehicle-side after a network delay in the downward[6] direction. Similarly to the upward case, the commands differ. In contrast, Figure 1.4c illustrates an ideal case, where networks impose no delays. The scene on the vehicle-side perfectly corresponds to the scene on the server-side. Likewise, commands applied to the vehicle are an exact match to the ones sent from the server. The contribution of this thesis, or the "big picture" is illustrated in Figure 1.4b. We make a hypothesis that our solution is a step towards the "ideal case". More precisely, we argue that both approaches outperform traditional implementations equivalent to the proposal in [6]. Although our solution is far from perfect, it is a step in the right direction, as the subsequent chapters show.

1.4 Outline

The remainder of this thesis is separated into six chapters.

Chapter 2 describes the concepts of machine learning, which form a base of our project. We emphasize on neural networks as the main point, with PCA as an auxiliary tool to improve the overall performance of our approach. Further, the state of the art approaches that tackle problems similar to ours are discussed. Finally, we discuss other approaches that tackle remote driving.

Chapter 3 provides a detailed recap of all software and hardware components, that were used throughout the project.

[5] From client to server.
[6] From server to client.

Chapter 4 illustrates one of our concepts for tackling network latency. Here, we proposed a system for predicting the future movement of all traffic participants that surround the remotely operated vehicle. Furthermore, in this chapter, we give a detailed implementation description. Furthermore, a thorough evaluation and discussion are presented at the end of this chapter.

Chapter 5 describes the approach for compensating network latencies, based on predicting commands that are not yet received. Apart from the conceptual description and implementation details, at the end of this chapter, we provide an evaluation between using and not using predictions when performing remote driving.

Chapter 6 contains the low-level details about the proposed solution and how it is used in the global environment. The descriptions contained in the chapter 6 serve as documentation to the implementations and the reasons behind them. We hope that this documentation will be the preliminary work to future improvements.

Chapter 7 summarizes the thesis with a discussion on the overall performance of both proposed approaches. Furthermore, chapter 7 describes the problems that occurred and gives an outlook about future work.

2 Fundamentals and Related Work

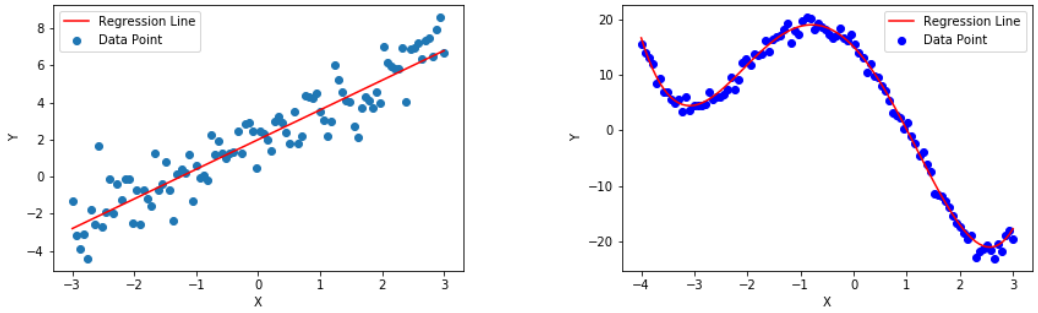
This chapter provides the reader with some insight into the concepts which are relied upon in the remainder of this thesis. We begin with the light introduction into the abstract idea of machine learning. In section 2.1, we introduce deep neural networks as well as their variants with feedback loops. They are used in this thesis in the sense that the complete approach that we propose is reliant on neural networks. Following are the regression methods, namely linear and polynomial regression. They are briefly described in section 2.1.1. The Positional Prediciton Approach implements one of its variants using regression methods to estimate future trajectories. Further, we describe the concept of Principal Component Analysis, which we use to reduce the dimensionality of our image-related problem, thus improving the proposed solution. Although it is not necessarily the centre of our interest, we use it as an auxiliary tool for improving the performance of the proposed Tele-Operated driving approach. Therefore, we dedicate section 2.1.2 to the PCA. In the subsequent section 2.2, we provide a broad overview of all papers and work that directly or indirectly influences our work. Most notably techniques about trajectory planning and modern approaches in using neural networks for predictions. The final section 2.3 of this chapter serves to draw parallels between our proposed solution and existing work in the field of remote driving.

2.1 Machine Learning

Described as *"The Computers ability to learn with data, without being explicitly programmed"*, Machine Learning is currently the most common approach in designing navigation systems for autonomous vehicles. Thomas Mitchell postulated the following:

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."[13]*

This proposal suggests there are both implicit and explicit ways of solving problems. These two concepts of learning are fundamentally different. Implicit learning is the learning of complex information in an incidental manner, without awareness of what has been learned[14]. In contrast, explicit learning corresponds to providing a solution for each possible instance of a given problem. When applied to the case of traffic behaviour, one can effortlessly conclude that the space of all possible scenarios, in which the vehicle may find itself, is immense. Therefore, we decide to tackle the problem of Tele-Operated driving in an implicit manner. Given recent advancements in this area of science, we decide to build and train a neural network and to exploit the generalization possibilities



(a) Illustration of linear regression.

(b) Illustration of polynomial regression.

Figure 2.1: Regression methods applied to the set of artificially generated data.

to be able to make predictions about vehicle positions and commands. Furthermore, we rely on techniques like regression and PCA to make the solution more comprehensive.

2.1.1 Regression

Regression is a powerful method for establishing a relationship between dependent and independent variables. In other words, regression is the statistical mechanism that minimizes the ordinary least squares error function. The core of regression methods is to fit the polynomial curve of some degree to the set of data points so that the distance between the points and the features is minimal, as illustrated in Figure 2.1. Generally, we distinguish between polynomial and linear regression, although polynomial regression is a generalization of the linear regression to the higher polynomial order.

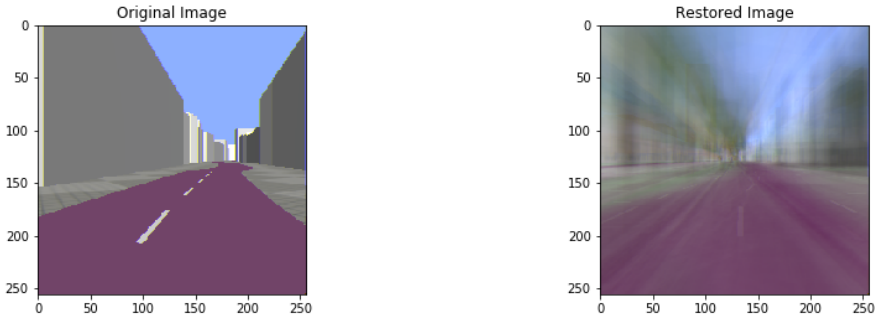
In the context of Tele-Operated driving, regression methods enable estimating future vehicle behaviour. A polynomial curve is fit to the set of previous consecutive positions. The newly fit regression line represents a trend of positional change during time. The future position estimation is available directly from the regression line.

2.1.2 Principal Component Analysis

Principal Component Analysis is a linear technique that is used in applications for data preprocessing, dimensionality reduction and data compression, feature extraction and data visualization.[15] In this work, we use PCA as a tool for reducing the dimension of image data, that is acquired by the vehicle. In general, PCA algorithms find a lower-dimensional projection for the data such that the maximum amount of variance is retained. Motivated by this idea, we aim to reduce the number of input dimensions for images, created by an on-board camera.

An example of the camera image can be seen in Figure 2.2a. The height and width is arbitrarily chosen as 256, which means that the overall RGB image dimension is 196608.[1]

[1]The number of dimensions corresponds to the number of different pixel values (65536), multiplied by



(a) The example image from on-board camera sensor, mounted on a vehicle in PHABMACS simulator.

(b) A reconstructed image, that was projected on 44 principal components and retains more than 80% information.

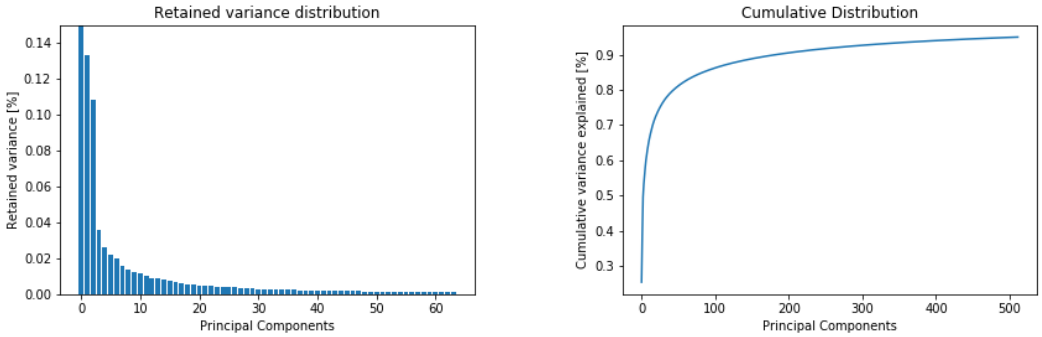
Figure 2.2: Comparison between the the original 196608-dimensional image, and the 44-dimensional image projection. Although humans may find the image 2.2a easier to understand, neural networks find the projections such as Figure 2.2b as a better foundation for successful generalization.

Unfortunately, such a high number of dimensions is too high in practice because it requires a complex model to successfully generalize[2]. Given that the driving decisions have strict timing requirements, we compromise between accuracy and computational intensity to enable real-time execution. Each image can be seen as a point in a 196608-dimensional space. At consecutive time intervals, we sample a camera sensor to capture the current image, thus creating a data set of driving images. Further, we computed a set of mutually perpendicular principal components that form a subspace, onto which each image point is projected, such that the direction of the principal components maximizes the data variance. To retain as much information as possible, we select a subspace, comprised of the first 44 principal components. By analyzing the contents of Figure 2.3, one can interpret the amount of information as the amount of retained variance after projecting the data onto the principal components subspace. Further, the cumulative explained variance threshold is selected as 80%, which means that the number of principal components is increased until the 80% of the information is contained in the feature space.

Figure 2.3 depicts the individual and cumulative contribution of each principal component. Figure 2.3a describes how the overall contribution of each new direction in the feature space declines as the number of directions increases. Intuitively, each projection direction is chosen to maximize the data variance. A further constraint is that each principal component is perpendicular to all others. Therefore, the lower principle components have fewer constraints to satisfy and therefore, can more optimize towards maximizing variance. Figure 2.3b describes the cumulative information retaining. By

the number of channels (we use RGB colour images).

[2]Predict unseen problem instances equally good as seen instances or with tolerable degradation.



(a) The amount of variance, retained by each principal component.

(b) Cumulative distribution function of all principal components.

Figure 2.3: The process of selecting the feature space that retains a significant amount of variance is a trade-off between the loss in the informational sense and gain in the lower dimension sense.

determining the threshold value, one can estimate the number of needed principle components, to be able to represent the data accurately.

Finally, we can represent each image with image 196608 dimensions as a 44-dimensional vector by projecting the image onto space spanned by the most significant principal components. By performing projections, we lose less than 20% of the original variance. In turn, we gain a set of vectors that are suitable for neural network inputs. Our initial results show that neural networks can generalize and converge to the optimal set of parameters with image projections. The benefit of using image projections instead of complete images is that image projections require a lower number of parameters which scales well with the real-time execution constraints.

2.1.3 Neural Networks

Neural networks are a class of machine learning frameworks that gained significant importance both in academia and in the industry due to their rapid performance increase. They are suitable for performing tasks that are "learnable". In other words, a supervised version of "implicit learning" corresponds to providing a set of inputs and target outputs to the model with specific learning mechanism. Upon the training procedure, the neural network "learns" how to mimic the provided behaviour by updating the model parameters to generate output values as close as possible to the provided targets.

Neural networks are complex models build by the interconnection of simple units. Such units, neurons, were first introduced by McCulloch and Pitts are carefully designed to mimic the human brain. The mathematical equivalent to this biological concept can be seen in Figure 2.4. The output of a neuron is dependent on the outputs of the neurons that are connected. Given two neurons, A and B, the correlation between neurons measure the dependence of the output from neuron A, given neuron B. This dependence

is reflected in the weighted parameter connecting neurons A and B. In other words, the output of neuron A is a function of a weighted sum, of the outputs from all neurons that are connected to the neuron A.

In mathematical terms, we write the output of a neuron as a function of its inputs and weight vector, that represents the connections to other neurons:

$$f(\mathbf{x}, \mathbf{w}) = \phi(\sum(x_i \cdot w_i) + b) \tag{2.1}$$

where $\phi(\cdot)$ is some nonlinear function. The graphical illustration is provided on Figure 2.4

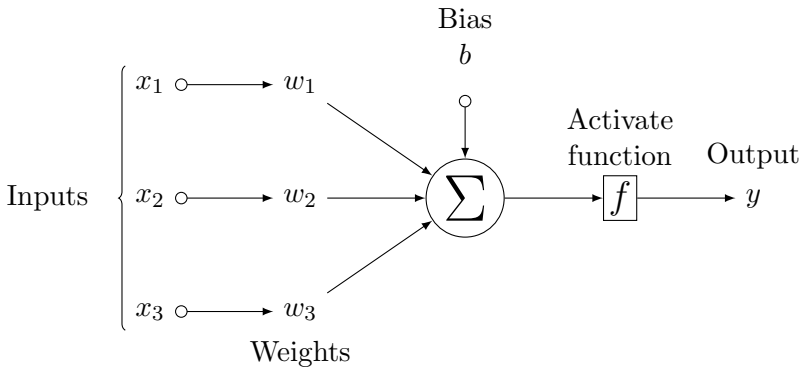


Figure 2.4: An example of a single artificial neuron neuron.

Neural networks are in general, formed by interconnecting neurons in layered structure as the one depicted in Figure 2.5. We identify the structure of a neural network by a sequence of layers, connected by weight vectors. By adding more hidden layers to the network architecture, we get so-called "deep neural networks". In general, the input-output relationship is learned by setting the weight parameters between certain neurons to the values which minimize the difference between the network predictions and target values. The algorithm for training neural networks is called "gradient descent"[15].

Some problems have correlation between instances in a sense that, given instance X_i, instances $X_{i+1}, X_{i+2}, X_{i+3}$,...$X_{i+n}$ are somehow dependent. To further exploit the temporal correlation between the inputs, the concept of feedback loops, present in the network architecture may impose itself as a useful feature.

> "Recurrent neural networks (RNNs) were originally developed as a way of extending neural networks to sequential data. Because of their recurrent connections, RNNs are able to make use of the previous context." [16]

Recurrent neural networks are the result of adding a cycle to the network graph, as shown in Figure 2.6. The primary motivation is for introducing cycles is to enable the dependency between neuron outputs between different instances, instead of relying solely on the dependence between the neurons in previous layers.[17] Figure 2.6 depicts

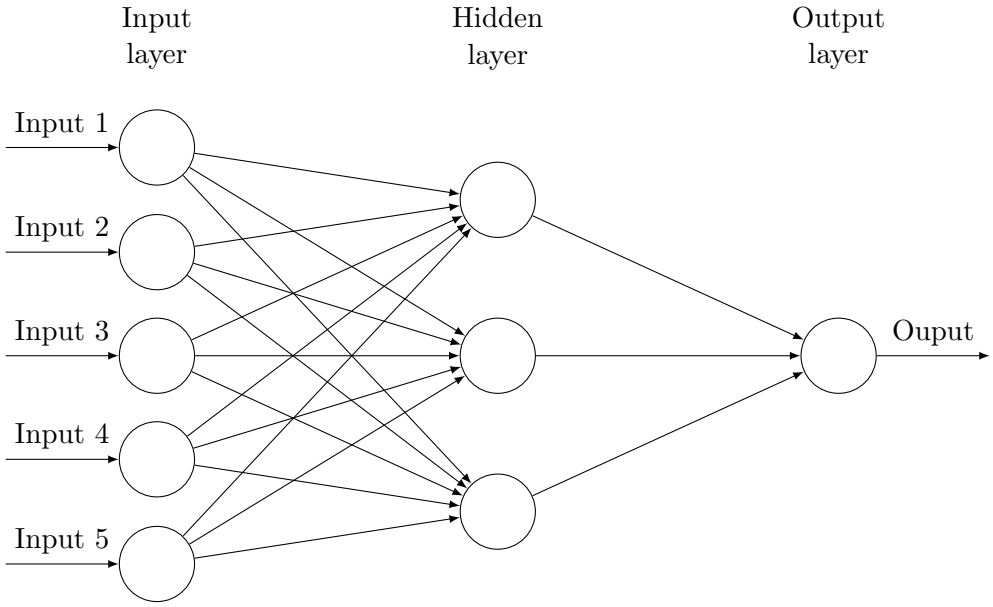


Figure 2.5: Example diagram of a neural network, comprised of multiple neurons, mutually connected into distinguishable layers. One can identify the input, the hidden and the output layer to the neural network.

a network containing neurons with feedback loops. If a network was driven first with input X_n and then with input X_{n+1}, when producing the output, for instance, X_{n+1}, neurons with feedback loops take in account their output when generating the output, for instance, X_n, while other neurons have no notion of temporal dependency and are fundamentally limited only to the current input instance.

Unfortunately, training RNN networks is a daunting task. Unlike the network without cycles, there is a correlation between neurons in the temporal domain. Additionally, the correlation between other neurons in the previous network layer is present. Training RNN networks is addressed as the problem of unstable gradients by [17]. The gradients that are propagated through many stages ether take insignificant values that practically do not change the weights between neurons, or take high values that lead to parameter instability. For circumventing the issues of training recurrent neural networks, Sepp Hochreiter and Jürgen Schmidhuber proposed a novel network architecture[18], which is proven to learn faster and with higher generalization performance than its RNN counterparts. Furthermore, upon introduction, LSTMs were state-of-the-art models that solved tasks that no other recurrent architecture solved at the time.

The fundamental reason behind their excelling performance when predicting sequences is the feedback loop that uses previous outputs as inputs for predicting the current output. This allows the information to persist[19]. In other words, past frames contribute

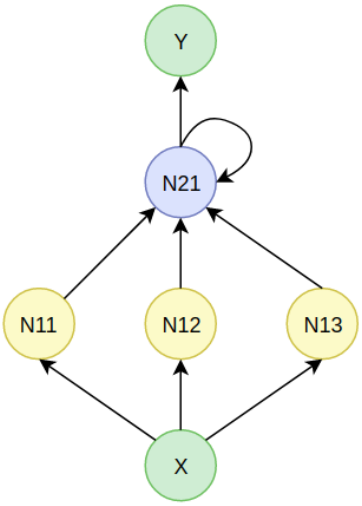


Figure 2.6: An example diagram containing two-layered network. First layer consists of three neurons, while the second layer consists of one neuron with a cycle.

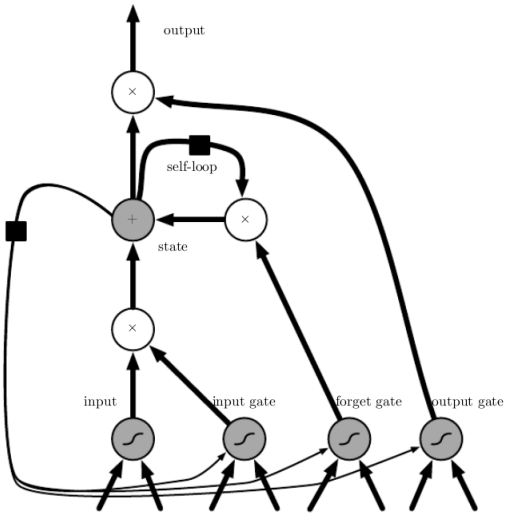


Figure 2.7: The block diagram of LSTM cell from [3], describing the data flow from input to the output, functionality and interconnection between different gates

to a better understanding of the current frame.

Figure 2.8 depicts the LSTM[3] cell, a variation of regular RNN cell. The main idea is to replace the fixed self-loop weight with context-dependent weight. This is particularly useful when predicting long sequences, where regular RNNs often tend to fail. Three gates control the information flow as they regulate which information goes through.

- Forget Gate - regulates how fast the network forgets previous states.
- Input gate - decides which values are going to be updated.
- Output gate - regulates the output of a network.

2.2 Related Work

This section should provide a broad overview of all papers and work that directly or indirectly influence our work. Most notably techniques about trajectory planning and modern approaches in using Neural Networks for predictions.

Although using DNNs in the field of autonomous driving and trajectory estimation is relatively young, its application is growing with significant research done in this field.

An approach similar to ours is shown in [20], where authors solved a regression problem of predicting trajectories using LSTM neural network, with RMS error of 0.7m for

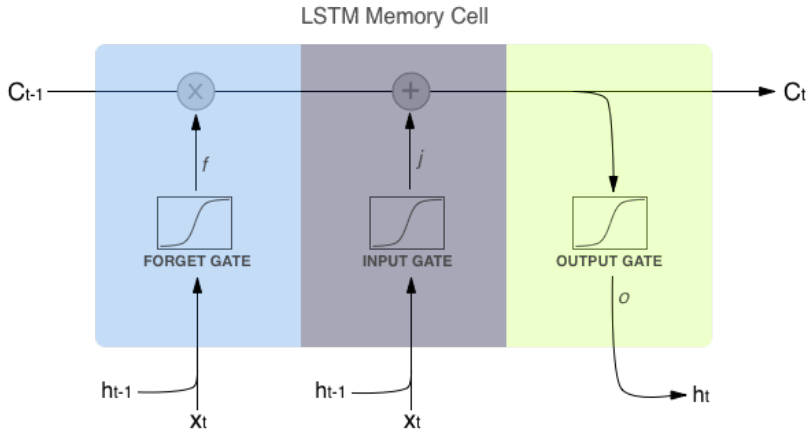


Figure 2.8: Illustration of the LSTM cell. **C** corresponds to cell output, **x** corresponds to cell input wile **h** corresponds to the state of a cell

predicting long-term lateral vehicle position. The problem that it tackled was highway driving, based upon NGSIM[21] dataset. The paper [20] served as a useful reference point throughout the development phase of our project.

Instead of a supervised approach, autonomous driving via reinforcement learning is used in [22] and [23]. Motivated by the performance on games such as Atari and GO, these papers scale up this approach to the problem of autonomous driving. The paper [23] proposes using 3D simulations for solving the problem of vehicle state transitions via Markov Decision Process, where the agent traverses through the set of states by performing actions [3]. Further, [23] introduces hybrid CNN-RNN agents that perform well for image-form input data. A similar approach is demonstrated in [22]. The downside of these approaches is that they are concerned with simulation datasets instead of real-world scenarios. Further, for the model to perfect generalization performance, the agent needs to explore all possible states, some of which may be dangerous for the driver. Therefore, this approach scales far better in simulations than in real-world scenarios.

An end-to-end learning approach is introduced in [24]. It relies on predicting the steering angle based on the image input, using a hybrid C-LSTM[4] neural network. The fundamental idea behind this approach is to exploit both spatial and temporal dependencies. First is done by performing convolutions, while the second is done through recurrences. Likewise, [25] also incorporates 3D convolutions, that are followed by recurrent layers. The network input is a set of the Udacity dataset images. The prediction is a continuous steering wheel angle.

Recurrent neural networks are also used in [26] and [27]. In both approaches, vehicle

[3]Actions consist of accelerating decelerating and steering
[4]Convolutional Long Short-Term Memory Recurrent Neural Network

coordinates are used for LSTM cells to produce a set of candidate trajectories for each of the surrounding vehicles. The candidate trajectories produce an occupancy matrix for each time step in the predicted sequence. In other words, the network suggests that a certain vehicle is going to be at a certain road location with some probability. Although this approach is exciting, the downside of predicting each vehicle trajectory with a separate LSTM network is not scalable for dense traffic. Isolating a vehicle from its surrounding is not practical, as the vehicle behaviour is very dependent on its neighbouring vehicles.

An interesting approach in posing trajectory prediction as a classification problem is demonstrated in [28]. In [28], vehicle behaviour is predicted at the intersection. The input to an LSTM-based model is a sequence, and the output value corresponds to one element in a discrete set of possible driver intentions.

2.3 Concurrent Approaches

There are significant contributions regarding Tele-Operated driving in terms of improving specific segments.

A paper [29] proposes a Tele-Operation concept and application. Apart from providing detailed structural architecture, this paper identifies the network latency as the main obstacle that needs to be mitigated. Furthermore, the paper proposes cellular networks with strict bandwidth requirements, suited for video transmission, as well as evaluates WiFi networks as having insufficient performance potential given mobility drawbacks. Finally, this paper suggests that driver awareness is a separate issue that needs attention. In other words, the issue with driver awareness is that a driver in a remote cockpit relies only on video transmission, while regular driving provides different sensory stimulation. Instead, only relying on visual information, the driver in a car relies on feeling the motion of the vehicle. Moreover, the driver is relying on the surrounding noise while driving.

In a PhD dissertation [30], the author identifies the lack of driver sensory input as one of the problems. If remote driving relies solely on video transmission, the driver in the remote cockpit has a minimal perception of the vehicle speed, in comparison to the regular driving. It is easily inferred that presence has a positive correlation with the quality of driving. If the remote driver lacks sensory information, there is a higher probability that the driving quality would decrease. To increase the "real-feel" of the driving, the authors propose optical, auditive and haptic feedback using blur. Furthermore, artificial motor sounds are proposed to increase awareness as well as vibrations at the driver's seat. Results, inferred from conducting studies show that augmenting information as proposed above increases the driver estimate of the speed.

The paper [31] investigate similar problems to the ones identified in this thesis. More precisely, the problem of mitigating the network delay between vehicle and server is mitigated by proposing various prediction techniques. These techniques are fundamentally based on deterministically computing vehicle position predictions from vehicle dynamics[5]. The results show that deterministic computations provide stable and accurate

[5]Vehicle dynamics refer to vehicle velocity, acceleration and position parameters

position predictions. Furthermore, given the real-time execution requirements, all approaches are characterized by sufficient execution speed.

The authors of paper [32] collect the data set comprised of driving recordings[6]. Based on the previously collected data, the authors conclude that, when performed on Mobile networks, Tele-Operated driving is feasible, with latency below 250 ms. Unfortunately, this study also found cases in which the overall latency was above 1 ms with high jitter and congested traffic, that leads to a decrease in the throughput. Particularly important is to handle the handover between cells.

[6]The driving recordings lasted 4660 minutes and covered 5200 km

3 Requirements

This chapter serves to illustrate the technical requirements that conditioned the development of our project. Although they do not contribute to the overall solution in a meaningful way, this chapter provides the formal requirements that were fulfilled. Section 3.1 states the hardware components that are used on this project, while section 3.2 provides the insight into software components. Finally, chapter 3 is concluded by the conceptual design section, in which conceptual requirements are covered.

3.1 Hardware Requirements

This master thesis is implemented in the Phabmacs simulator. Therefore, hardware requirements are relatively flexible. Low computational power is needed to run Phabmacs and therefore, a simple laptop, whose components are listed in Table 3.1, is used.

Component	Name	Description
Model	HP Laptop	15-bs1xx
Processor	Intel i5-8250U	Octa-core @ 1.60GHz
System Memory	DDR 4	8GB @ 2400MT/s
Storage	SSD	256GB

Table 3.1: The tabular overview of the hardware components.

Initially, to control the Tele-Operated vehicles, a laptop keyboard was used. Given that the discrete input of keyboard buttons is not similar to the real-world scenario, this input was used only for initial tests. For further development and testing, an Xbox controller is used. Custom designed drivers are developed to enable the functionality. Furthermore, an Xbox was equipped with a custom mini steering wheel to control steering better. The complete control input device can be seen in Figure 3.1. The rotation of the steering wheel is translated to linear movement on the X-axis of the left stick-button. Throttle and brake are mapped to left and right trigger buttons.

3.2 Software Requirements

The Table 3.2 provides an broad overview of the software components that this project mostly relied on.

Phabmacs is a driving simulator that is developed by the Fraunhofer Fokus. It is used for implementing automotive services and technologies. Furthermore, it is used for enhancing safety and traffic efficiency, as well as implementing state-of-the-art ADAS.

Figure 3.1: Xbox ONE controller, equipped with a MINI-steering wheel, that was used as a user input for generating driver commands.

Component Name	Version
Phabmacs	
Eclipse IDE	2019-03(4.11)
Java	1.8.0_212
Python	3.6.8
TensorFlow	1.13.1

Table 3.2: The tabular overview of the software components.

Eclipse is an integrated development environment that is used throughout this project to develop partial components of the Tele-Operation concept. These components are then simulated in Phabmacs simulator.

Java is a general-purpose, object-oriented programming language. It is used or interacting with the Phabmacs simulator. The main characteristics, robustness and high performance, are proven to be crucial for implementing the Tele-Operated driving applications.

Python is a programming language that is used within this project for:

- Data Pre-Processing
- Neural network design, training and evaluation

- Extracting neural networks as computational graphs.

. The man characteristics of Python are that it is an interpreted, high-level, general-purpose programming language. The reason behind relying on Python is the convenience of using different frameworks to realise different tasks promptly and successfully. The strongest argument for adopting Python approach is certainly designing and training neural networks, which would be significantly difficult to implement in Java directly.

TensorFlow is one of the many libraries that is used for machine learning tasks. Developed by Google Brain team in 2015, it grew in popularity and is currently one of the most commonly used libraries for tasks like neural network design. Its popularity and broad application use cases are the main reasons for employing TensorFlow.

3.3 Conceptual Design

This section provides a conceptual design of the thesis as well as the conclusive elaboration behind the thesis structure. Furthermore, in this section, we discuss the possible ways of circumventing this thematic and our reasoning why the issues presented in this work require solving in the way we propose.

In the remaining of this section, we will further describe the organization of this thesis, as well as the reasons behind it.

Although resolving network latencies may be directly solved by optimizing networks, we propose alternative methods that mitigate this issue on the application layer. We do so primarily to avoid relying on the networks for our system to function correctly, but rather to obtain the model that maintains high performance even in highly congested networks.

The contents of this thesis are split into two almost distinct chapters, chapter 4 in which we propose predicting position in order to compensate for the network delays. Similarly, chapter 5 suggests tackling the same problem on the server-side by predicting the unreceived commands. Naturally, the two approaches are mutually resembling and rely on similar underlying functionality. Therefore, information repetition is expected to some extent. Nevertheless, this work clearly distinguishes between the two approaches.

The primary reason behind the split between the two approaches is because they are conceptually different. One is performed on the server-side; the other is performed on the vehicle side. Furthermore, message contents are different since different information is sent and received. Moreover, the client and server operate differently for both approaches. Finally, the methods are evaluated by different metrics. The additional motivation behind the split was to keep the thematic of both approaches separate so that the following thesis workflow is straightforward. Furthermore, we try to avoid unnecessary confusion by referring to both methods in parallel. Although it might be evident for the scientists, that works extensively in similar fields to this one, to follow the work, the split between the approaches prevents the confusion and difficulty with keeping up with various dissimilarities between two methods.

The further question that imposed itself is a matter of evaluation of two approaches. Naturally, we aimed to compare them against each other in order to establish a clear

favourite. Furthermore, this would enable a concise and clear proposal for all Tele-Operated automotive applications.

During conducting research on this topic, we conclude that there are reasonable doubts behind direct model comparison. Our argument is that they don't include or exclude each other in any way (both of them can be used, only one of them, or none of them). Moreover, both approaches are implemented separately so that anyone that is interested in conducting further research may use any of them regardless of the other. Finally, we conclude that it is reasonable to evaluate both approaches by their metrics. In other words, the position prediction approach is based on predicting the position and direction of the vehicle. There are multiple implementations of the same method (motion vector, linear regression, polynomial regression, neural networks). Different models are evaluated against each other to see the accuracy disproportion, computational complexity and resiliency to the network delays and noise.

Command prediction approach is based on predicting commands, regardless of the fact if we perform position prediction on the server-side. One natural metric may be to evaluate how successfully the predicting system is when predicting commands(throttle, steering and brake) by comparing the closeness of the current controls applied by the human driver.

The separate evaluation metrics enable the possibility to combine the two approaches. For example, when using the command predicting approach from chapter 5 and polynomic regression from chapter 4 in parallel, they do not influence[1] each other since one is applied on the vehicle-side and one is applied on the server-side. Moreover, the evaluation results from sections 4.4 and 5.4 provide further insight into this thematic.

[1]Their interdependency is non-existing. Performance of one approach does not depend on the implementation and the performance of the other.

4 Position Prediction Approach

This chapter demonstrates various techniques for predicting vehicle positions and heading. By predicting future vehicle trajectories, we can compensate for the messages that are not yet received by the server, due to the network latency. In other words, we propose various methods for generating trajectory predictions. By relying on these predictions, the person sitting in a remote cockpit can visualize the current state of the vehicle surrounding, thus eliminating the inbound[1] delay. Furthermore, we can generalize this approach and successfully predict positions further into the future. Position prediction is particularly useful if we aim to display the scenery to the driver, that is an estimation of the scenery at the moment when the commands reach the vehicle. Ideally, this approach would adequately compensate the network latency, thus reducing the adverse effects of both inbound and outbound[2] delays.

4.1 Concept

To better grasp the proposal, let us rephrase the problem statement by the following :

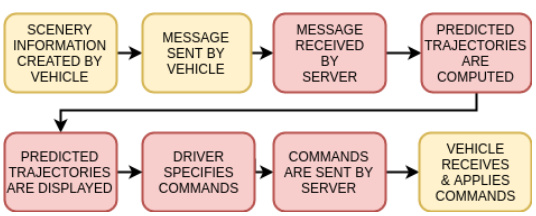
> *Develop a proof of concept prediction system for Tele-Operated vehicles, that predicts vehicle trajectories and compensates for the unreceived messages. These predictions should be based on the sequence of past information that is available. Such prediction systems should be able to accurately predict future trajectories(vehicle positions and heading) for a time interval that spans as further as possible into the future with satisfying performance.*

Using the above problem statement, we aim to propose a proof of concept prediction systems that tackle latency on the server-side for Tele-Operated driving. This section proposes a high-level architecture description and serves to introduce the concepts without going into too much depth. More detailed descriptions follow in chapter 4.3.

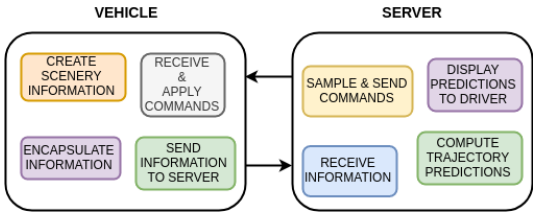
The diagram of the proposed approach is illustrated in Figure 4.1. The Figure 4.1a describes a set of most significant events that constitute the Positional Prediction Approach. The proposed illustration can be seen as a chain of events, executed on the vehicle- or the server-side Furthermore, Figure 4.1b describes the distribution of the main tasks among the vehicle and the server.

The Tele-Operated vehicle creates scenery information of all surrounding traffic participants as a list of virtual objects, each uniquely identifiable by the motion parameters:

[1]The communication delay between vehicle and the server
[2]Delay between the server and the vehicle



(a) Conceptual event chain diagram of the Position Prediction Approach. The events related to the vehicle are displayed in yellow, while the events related to the server are displayed in red.

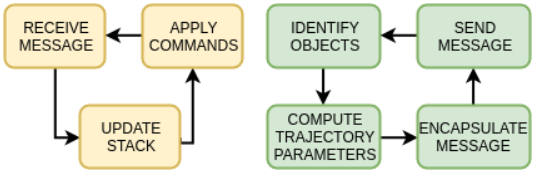


(b) Conceptual task distribution for Position Prediction Approach

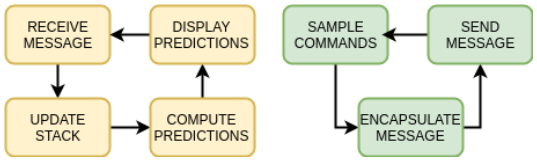
Figure 4.1: Task distributions for Positional Prediction Approach

- Position
- Velocity
- Lateral and longitudinal acceleration
- Front direction

The complete scenery information is encapsulated in a custom-defined message that is sent to the remote server. Given latency, that is introduced by the network; the server receives the message after some delay. Using the set of previous messages that are received from the Tele-Operated vehicle, the server can compute the positional prediction for each participating entity[3]. The prediction is displayed to the operator, who is responsible for generating the appropriate commands for a given scene. Further, the commands are returned to the vehicle. Finally, a set of most recent commands are applied to the vehicle.



(a) The task execution cycle for the vehicle when using Positional Prediction Approach.



(b) The task execution cycle for the server when using Positional Prediction Approach.

Figure 4.2: Task distributions and execution cycles for Positional Prediction Approach

To fully grasp the conceptual system execution, Figure 4.2 describes various tasks and their mutual dependence.

[3]Traffic entities are currently vehicles. Still, the concept is easily generalizable to different vehicle classes, pedestrians and other traffic objects.

Figure 4.2a describes the operation cycles that conceptually comprise the implementation of the Tele-Operated vehicle. It is essential to keep in mind that the operation is divided into two cycles, one handles sending updates to the remote server, while the other applies the commands received by the server. Both execution cycles are independent of each other, meaning that they can be implemented in parallel. The benefit of their mutual independence is that they may be performed on different hardware components. The further advantage of parallel implementation is that it increases the concurrent execution and reduces the timing constraints imposed by the real-time requirements. In other words, having a low number of small and independent execution cycles means that they increase the performance of the overall Tele-Operated vehicle.

The operation cycle of sending updates to the server can be seen in Figure 4.2a on the left side. To generate adequate information, the vehicle is responsible for identifying traffic objects. The motion parameters of these objects are then computed, encapsulated in a message and sent to the server. This completes the cycle of sending information to the server. The "command" cycle consists of the tasks that tackle the problem of receiving commands from the server. Upon reception, the message is added to the stack of previously received messages, that is arranged in the ascending order according to the send time. This set of methods keep the most recent messages on the top of the stack and handles message overtaking[4], that often occurs in the networks.

The operation cycle of the server is illustrated in Figure 4.2b. Similar to the Tele-Operated vehicle task cycle, the set of actions may be split into two sub-cycles to increase performance and get more concurrent system. The left part of Figure 4.2b, consists of the tasks related to displaying the information to the remote driver. The receiving of a message from the Tele-Operated vehicle is followed by updating a stack of messages to handle the possibility of out-of-order messages. This results in a sorted stack of messages which serves as a foundation for creating predictions. Using various prediction models, that are introduced in section 4.3, the information about previous trajectory parameters for all participants is used to generate future position and heading direction. These estimations are then displayed to the driver. The second cycle on Figure 4.2b relates to generating and sending the commands from the server to the vehicle. As a result of the displayed information, the driver generates a set of commands. These commands are then sampled, encapsulated in a message and sent to the vehicle, thus completing the server-related set of tasks.

4.2 Methodology

This chapter is used to describe the processes that are auxiliary to the implementation. Given that the most significant time was dedicated to training and evaluating neural networks, as well as importing them and running in Phabmacs simulator, this section provides more information about the project flow between conceptual planning and integrating a given component into the solution.

[4] As a result of using different routes; messages may arrive at the destination in the reverse order to the one in which they are sent. This phenomenon is called message overtaking.

The neural networks are the core component of our contribution. They run within the Phabmacs simulator to compute future positions and directions of all traffic participants.

To use the neural network components, the computational graph, that is externally created, has to be imported into the simulation. The term "computational graph" corresponds to the set of operations using predefined network parameters[5]. As the name suggests, it is a dataflow graph that is equivalent to the trained network. The nodes correspond to the operations that are executed during the forward pass, while the weights of the graph correspond to the inputs and outputs to those operations. We ran specific tests and comparisons to make sure that the network behaves the same way in Phabmacs simulator, as in the external Python scripts that created the computational graph.

The creating and evaluating of the neural network graphs is done externally in Python using TensorFlow, where the model architectures are built. Apart from the model architectures, the training procedure is designed to converge to the optimal set of parameters, which would enable the model to predict well.

To perform network training, we created a training set as a CSV file that consists of the vehicle position and direction in consecutive time steps. We derive this file by sampling the simulation in which the user controls the vehicle. The sampled values are added as rows to the external file. The resulting file is tabular-like trajectory information that extends for a particular driving period. The goal of the training is to select a batch of random samples and a target prediction and to minimize the difference between the network output and the actual value.

To keep track of the network performance, we derived a test set, similarly to the training set. Although the creating procedure was similar, the actual driving was done by another driver. Changing the driver contributes to the generalization performance of the network. In other words, we aim to have a model that predicts the general driving concepts, rather than the driving of a single person.

The result of the training procedure is a neural network with a converged set of parameters in the form of a computational graph. This graph is stored externally, to import it in the Phabmacs simulation.

4.3 Architectural Design

This section contains an overview of the system architecture as well as descriptions for all relevant components, whereas more detailed information may be found in 6. Apart from auxiliary components that serve to enable Tele-Operation and communication between the vehicle and the server, we introduce different mechanisms of generating predictions.

4.3.1 Auxiliary Components

The core component of any Tele-Operation system is the network that enables the communication between the vehicle and the server. Three following components can model it:

[5]By term network parameters; we consider weights, biases and activation functions

- Network Delay
- Message
- Message Stack

The network delay is an essential part of the communication. As the Tele-Operated vehicle and server exchange messages, the network delay is the system component that is responsible for the amount of the outdated information. In other words, network delay causes the server information to be outdated as well as the commands sent to the vehicle. The main characteristics of the network delay are that it is variable. Notably, it can oscillate between different values, depending on network congestion, signal strength, etc. This variability implies that message overtaking can take place. In our Tele-Operation use case, this means that a vehicle position seen by the server may appear in reverse order to the real scenario. Although negative, this property of the network delay must be tackled. We implement network delay as a parameter that is sampled with each message sending from a gaussian distribution. This enables us to genuinely control the average delay, as well as the amount of "spread" by adjusting the function parameters[6].

A message component is an object used for communication between Tele-Operated vehicle and server. The common parameters that constitute the message are:

- Target is an object to which the message is sent. When performing reception, the target adds the message to the Message Stack Component, after which the stack component is sorted.
- Send Time is the parameter that corresponds to the time when the message was encapsulated. The purpose of having this parameter is that the abovementioned stack sorting is performed for this parameter.
- Receive Time is a parameter which incorporates communication delay into the system. The latency is added to the send time, this creating a receive time. The messages are available for reception only after the simulation is progressed past the receive time. Afterwards, the messages may be added to the stack of the target.

Additional message parameters are dependent on whether the message originates from the vehicle or the server. If the vehicle sends a message to the server, then the purpose of the message is to contain the trajectory information for all surrounding traffic participants. This is done by sending a list of objects where each entry corresponds to one vehicle. The objects contain trajectory information, namely, unique identifier, position, lateral, longitudinal acceleration and heading. In contrast, if the message is sent to the vehicle, it contains a set of primary driving parameters. More precisely, steering angle, throttle and brake.

Message stack is the component that is responsible for keeping track of the most recent messages. Implemented as a list of messages, message stack adds a new message to the

[6]Parameters of the gaussian function are mean and variance

list followed by the sorting mechanism. As a result, the messages are always sorted in an ascending way with respect to their send time. Fetching a set of most recent messages corresponds to taking messages from the top of the stack, which proves convenient when using a set of messages to generate predictions.

4.3.2 Prediction Mechanisms

The core contribution of this work is introducing prediction mechanisms that enable reliable trajectory computing. So far, we have designed multiple prediction mechanisms to compare and evaluate the performance of each one.

- Most Recent Parameters - Is the prediction mechanism where no prediction is made. Although the term "prediction" is not applicable, we use this method to demonstrate how outdated the information is without predictions. It serves only for comparison purposes. Fundamentally, it is implemented as using most recently received set of parameters (position and heading). The comparison with other mechanisms serves to demonstrate the necessity for trajectory estimation. The motivation for using this prediction mechanism is to have a benchmark that corresponds to the systems described in [6].

- Last Motion Vector - Is the prediction mechanism that relies on the difference between two consecutive position and heading measurements. The difference between them is divided by the time difference, resulting in a parameter of the increase/decrease per time unit. Assuming this parameter is constant, we may be able to compute the expected positional and direction change for an estimated delay. Motivations for using this approach are its light weightiness and negligible computational complexity. Therefore, it is applicable to almost any underlying hardware platform.

- Ridge Regression - Is the prediction mechanism where we assume a linear correlation between trajectory parameters. A mode detailed description is provided in section 2.1.1. The set of previous positions may be seen as a set of 3D vectors in a 4D space, where the additional dimension is the time corresponding to each position vector, the ridge regression finds parameters of a linear approximation which fits the positional parameters. The hyperparameter lambda is a regularization parameter, adjusted to minimize the negative effects of outliers and reduce overfitting. The motivation behind using regression is simplicity and the fact that a trajectory may be approximated with satisfying accuracy with a set of linear segments.

- Polynomial Regression - Is an extension of the Ridge regression to using the polynomial function to fit the position and heading vectors, rather than a linear model. The motivation behind this approach is to increase to approximation possibilities. In turn, the computational complexity of using these models is increased. We may see this as a trade-off between accuracy and complexity.

- Deep Neural Network - Train an LSTM neural network model implemented in TensorFlow to obtain a model with satisfying accuracy. Invoke this model in Phabmacs simulator to obtain predictions of future positions and heading parameters. We designed and trained a Deep Neural Network consisting of three LSTM layers, each with 64 neurons. The motivation for conducting experiments using deep neural networks comes from state of the art results [20] in the field of trajectory estimation.

- Deterministic Prediction - Is a prediction mechanism that relies on computing trajectory components using the most recent position, lateral and longitudinal acceleration and forward direction. This prediction mechanism is developed independently and externally from this thesis by a colleague in the DCAITI. We use it as a benchmark for accuracy because of its comprehensive nature and speed because it is executed directly in the java code without APIs.

4.4 Evaluation

This section serves to provide the cross-comparisons of different prediction mechanisms. It offers essential arguments and numerical justifications for the usage of predicting positions when tackling latency issues in Tele-Operated driving. After describing the evaluation methodology at the beginning of this section, we provide evaluation results and compare the scenarios in which different prediction mechanisms are used to determine the position estimations.

4.4.1 Test Environment

The tests are carried out in Phabmacs simulator. After designing the Tele-Operation scenario, we model the scenery that is displayed on the server-side. Figure 4.3 depicts a screenshot of remotely operating a vehicle with the aid of the predictions. The remote operator is displayed with the predicted positions, corresponding to the time, by which the commands reach the vehicle. These predictions are displayed as the blue bounding boxes.

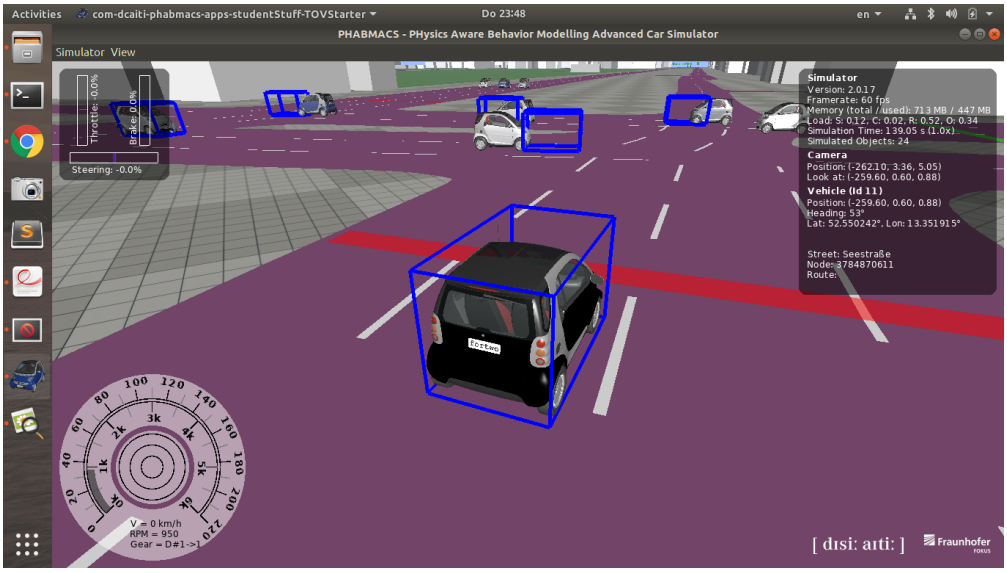


Figure 4.3: A screenshot of the Tele-Operation scenario in which controlling a vehicle is aided by the positional predictions.

4.4.2 Scalability

The scalability of the solution is a crucial aspect that describes how the solution scales with the increase of the required work. If this is projected onto the problem of trajectory estimation, we see that scalability is measured according to the increase of the vehicles whose trajectory is to be estimated. In other words, the more vehicles we take into account, the more predictions need to be computed. To fully satisfy the real-time requirements, all predictions must be computed in a timely fashion. Therefore, we measure the scalability of the solutions by comparing the required time to produce an estimation. The same platform and evaluation environment is used to cross-compare different prediction models, and therefore the reflected differences are a pure measurement of their algorithmic complexity. The Table 4.1 describes the time per execution of different prediction mechanisms. It is measured in micro-seconds by a callback function that compares the timestamps prior and after a prediction generation function. Further information about the exact implementation is contained in section 6.6

Approach	Execution Time - Mean	Execution Time - Standard Deviation
Ridge Regression	986.9535	356.2903
Polynomial Regression	1394.8462	579.8376
DNN	42772.6030	3211.6121
Deterministic	6.8320	7.3571

Table 4.1: The tabular overview of the average execution time associated to different approaches.

The values in the table 4.1 represent the execution time for different approaches in the Phabmacs simulations. The measurements are shown in microseconds. The underlying hardware significantly differs when some of these approaches are finally applied in automotive applications. Therefore, numerical values are not as important as the difference in the computational efforts required to complete predictions successfully. An approach for predicting positions based on the neural networks is significantly ineffective in terms of computational complexity. Our models show a lack of ability for fast and lightweight computations, in contrast to polynomial and ridge regression models. Further, a deterministic trajectory computation model, reliant on the vehicle velocity achieves the best performance.

4.4.3 Usability

We measure usability by evaluating how different prediction mechanisms manifest their performance. The interesting factor is the prediction variance between two consecutive predictions. We call this factor the jitter. Although the term jitter is used in various scopes, none of which we adopted it adopted from, our interpretation of the term and the metric is simply a positional difference between consecutive camera standpoints. In mathematical terms, jitter is a subtraction between vectors that describe the camera positions. High jitter reflects in a great difference between consecutive predictions, while low jitter corresponds to more similar predictions. Naturally, low jitter is desirable. We evaluate the jitter by adding measurement noise to the actual parameter measurement[7] The motivation for adopting a metric like jitter to evaluate usability is to translate the prediction uncertainty and variation in the output into the users perspective. If the prediction mechanism exhibits strong oscillations, they reflect in oscillating the visual perspective on the server-side. This is justified by the fact that we may not be able to measure the positions of all objects in the scenery perfectly, as well as the position of the ego vehicle. Table 4.1 describes the comparison between different approaches with respect to the average jitter. This reflects the difference between consecutive estimations.

Approach	Average Jitter	Jitter Variance
Ridge Regression	0.0054	0.0089
Polynomial Regression	0.0130	0.0200
DNN	0.0227	0.02968
Deterministic	0.0144	0.0158

Table 4.2: The tabular overview of the average jitter parameter associated to different approaches.

Since Phabmacs position vectors use meters, thus the values in table 4.2 are in meters. Although the difference numerically seems negligible, similarly to the strong hand

[7]The noise is to the position, acceleration and forwards direction of all vehicles when the ego-vehicle measures these parameters.

shaking when recording videos, the pleasant visual feel is quite different for various jitter results.

From Table 4.2, it is observant that the prediction models, based on ridge regression, achieve the lowest jitter values. In other words, a prediction model that is implemented as linear regression has insignificant positional oscillations. In other words, a Tele-Operated driving system runs the smoothest, if implemented using linear regression. Following are the polynomial regression and deterministic predictor that achieve similar performance. Finally, the most significant oscillations between consecutive predictions are observable when using deep neural networks.

4.4.4 Performance Measurements

Solution accuracy is a significant parameter that indicates how much does a specific approach err when producing predictions. Particularly, we measure the extent of the difference between the actual positions and the predictions. Methodologically, we achieve this by sampling the simulation and deriving the estimated and actual positions and vehicle heading. This information is afterwards stored in external CSV files. Finally, we design scripts that automate the evaluation process that is presented below. The difference between different prediction approaches regarding accuracy may be seen in Table 4.3.

Approach	Average Error	Error Variance
Ridge Regression	2.2743	1.2252
Polynomial Regression	1.9224	1.1094
DNN	4.0384	2.3291
Deterministic	1.6405	0.6796

Table 4.3: The tabular overview of the average position error parameter associated to different approaches.

The table 4.3 contains the accuracy performance results when prediction vehicle positions. Here, the deterministic approach achieves the best results, followed closely by the regression models. The neural-network-based prediction mechanism achieves low accuracy for predicting positions.

Approach	Average Error	Error Variance
Ridge Regression	0.0763	0.1161
Polynomial Regression	0.1890	0.1165
DNN	0.4119	0.1611
Deterministic	0.0530	0.0765

Table 4.4: The tabular overview of the average error direction parameter associated to different approaches.

When vehicle heading is a feature that is predicted, the same relationship between the approaches is maintained as when predicting position. The reason behind this is that predicting direction is similar to predicting location. More precisely, a deterministic approach outperforms other approaches, with regression methods achieving similar results.

5 Command Prediction Approach

This chapter demonstrates a machine-learning-based technique for predicting vehicle commands. By estimating future[1] commands, it is possible to compensate for the messages that are not yet received by the vehicle, due to the network latency. In other words, we propose an approach that incorporates recurrent neural networks for generating command predictions. By relying on these predictions, the vehicle applies the commands that are more similar to the ones currently applied by the person sitting in a remote cockpit, than the most recently received ones. Command prediction is particularly useful if the aim is to correct the disposable commands. Ideally, this approach would adequately compensate the network latency, thus reducing the adverse effects of outbound[2] delays.

5.1 Concept

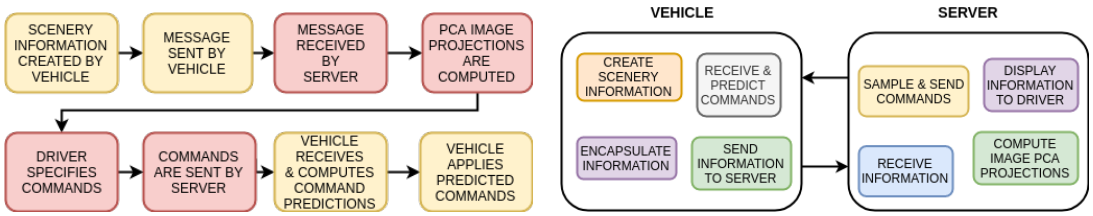
To better grasp the keynote of this proposal, let us rephrase the problem statement by the following :

> *Develop a proof of concept prediction system for Tele-Operated vehicles, that predicts unreceived commands based on the sequence of the past sensory information and their corresponding commands. In other words, relying on the camera images and the corresponding commands, enable predictions of the commands that are not yet received. Such predictor should be able to accurately predict primary driving parameters (steering angle, braking, acceleration) for a time interval that spans as further as possible into the future, with satisfying accuracy. The predictor should be trained on a training set, consisting of sensory information and commands that the model should predict. An evaluation should be done on a test set, in which the model uses a sequence of commands and sensory information to predict a future trajectory.*

Using the above problem statement, we aim to propose a proof of concept prediction systems that tackle latency on the vehicle side for Tele-Operated driving. This section is a high-level description chapter which serves to introduce the concepts without going into too much depth. More detailed descriptions can be found in chapter 5.3.

The diagram of the proposed approach is illustrated in Figure 5.1. Figure 5.1a describes a set of most significant events that constitute the Command Prediction Approach. The proposed illustration can be seen as a chain of events which correspond

[1] Commands that are sent, but not yet received as a result of the network latency
[2] Delay between the server and the vehicle



(a) Conceptual event chain diagram of the Command Prediction Approach. Events related to the vehicle are displayed in yellow, while the events related to the server are displayed in red.

(b) Conceptual task distribution for Command Prediction Approach

Figure 5.1: Task distributions for Command Prediction Approach

to the vehicle or the server. Furthermore, Figure 5.1b describes the distribution of the main tasks among the vehicle and the server.

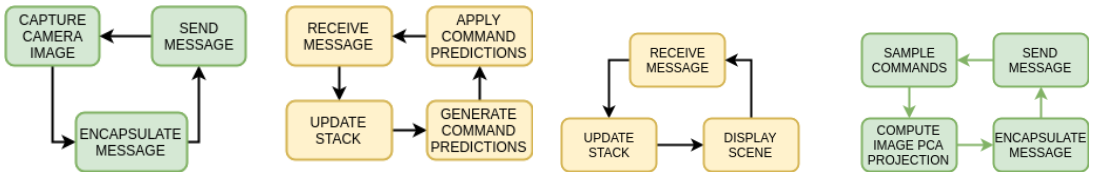
The Tele-Operated vehicle creates scenery information in the form of capturing camera images of the surroundings. The images are sent to the server where they are displayed to the remote driver. Based on the images, the driver generates a set of primary driving commands[3] that are sent to the vehicle. The server-side is responsible for computing the PCA projections of the images. Furthermore, this information is incorporated into the outbound message as an addition to the commands. Finally, the server sends the message to the vehicle, where it is received with a given delay. Upon receiving a delayed message that contains a set of commands and the image that corresponds to these commands, the vehicle is responsible for maintaining the message stack in sorted order according to the message send time. Using a set of most recent messages that contain commands and corresponding images, the vehicle uses pre-trained neural networks to generate a set of predictions. These predictions should correspond to the commands that would have been received given zero-delay. Finally, the neural network predictions are applied to the vehicle.

In comparison to the predicting vehicle directions and positions, for predicting commands, we aim to utilize images as an auxiliary source of information. Although during our work, we were constrained both by the lack of data and processing power, we were determined to extend the information source to include the surrounding scenery, instead of just a set of consecutive commands. Both the lack of data and processing power limited the set of feasible solutions that included using state of the art convolutional networks. Instead, we have decided to use low-size image embeddings. In other words, we use a subset of Principal Components and project an image to the space spanned by the PCs in order to compress the information on the vector of relatively small dimension. The motivation for using Principal Components of an image came from Olivetti faces dataset[33]. A common application is to represent facial features by only a subset by

[3]Without the loss of generality; we consider throttle, brake and steering wheel angle a set of fundamental driving commands.

carefully extracted information. Further encouragements to our work were the image-embedding applications for the PCA, as shown in [34]. The authors proposed embedding a high-resolution image, to only a few principal components that retain large amounts of variation. The reconstructed images were highly similar to the original ones. Furthermore transmitting lower amount of data is of vital importance to any network application such as Tele-Operated driving. Therefore, we aimed to represent road information using fewer parameters as possible. For small datasets, as we had, low information quantity enables relatively good generalization. Since the initial experiments showed that generalization is possible, we tried extending our set of principal components to more than a hundred. Although we expected that additional information could have a positive contribution to the overall performance, we experienced hard overfitting. In other words, the networks relied only on the image data and not the set of previous commands when predictions were made. Furthermore, we hope that future work will alleviate data and computation constraints, and therefore, improve on our results using convolutional neural networks.

By further analyzing the example image from Phabmacs simulator on Figure 2.2a, we can observe that the image consists of pixels with relatively low colour variation. More precisely, grey and purple pixels are dominant since they represent buildings and road, respectively. Therefore, there is justifiable concern that the lack of variation in the colourspace of the input images jeopardizes the given approach. On the other hand, limited variation may be one of the causes for hard network overfitting that we experienced. This issue is further discussed in section 7.2.



(a) The task execution cycle for the vehicle when using Command Prediction Approach.

(b) The task execution cycle for the server when using Command Prediction Approach.

Figure 5.2: Task distributions and execution cycles for Command Prediction Approach

To fully grasp the conceptual system execution, Figure 5.2 describes various tasks and their mutual dependence.

The execution cycles on the vehicle side are displayed on Figure 5.2a. The camera images are captured and encapsulated into the message along with the parameters that uniquely identify the message. These parameters are sent time and server identifier. Further, the message is sent, and after a certain delay, received by the server. Upon receiving the message, the vehicle is responsible for updating the message stack by ordering the messages in ascending order according to the message send time. The resulting stack component holds a set of most recent messages and the message overtaking, that may occur is adequately handled. Before applying commands, the vehicle uses the set of most recent messages that contain generated commands and the corresponding PCA

projections and creates a set of commands that correspond to the commands that are currently applied by the server, thus eliminating the message delay. Finally, command predictions are used.

The execution cycle of a server component in our system proposal is displayed in Figure 5.2b. One part of the execution chain is responsible for maintaining the stack in order, according to the timestamps of individual messages. Displayed in yellow, this sequence of events starts with receiving messages. It is followed by ensuring that the out-of-order messages are adequately placed. The event chain ends with displaying the information to the remote driver. The second part of the cycle relies on the previously stacked set of messages. For each received message, a collection of primary driving commands are generated. The image as an auxiliary source of information, with which the commands are coupled. The image is projected onto a set of principal components and sent in addition to the commands. The motivation behind this approach is for the vehicle to establish a correlation between the image and the corresponding commands. Using this information, the neural networks, that are behind the prediction mechanisms are trained to generate future driving commands.

5.2 Methodology

This chapter is used to describe the processes that are auxiliary to the implementation. Given that the core of this approach relies on machine learning, here we propose approaches to the acquiring and manipulating the data for the neural networks. Furthermore, we define the formats in which the data is used.

Instead of being explicitly programmed, machine learning models are said to learn from data. In other words, using optimization algorithms such as backpropagation, neural networks converge to the optimal set of parameters while using training data. Therefore, the overall performance of the system is dependent on the quality of the data. This data is collected, manipulated and finally used during the training phase.

To derive the training data, that is later used to train and optimize our neural networks; the Phabmacs simulation was iteratively sampled. Consecutive periodic execution enables the opportunity to observe and learn time-correlated driving behaviour, that can be later learned and optimized by the neural networks. More precisely, on each simulation sample, the current camera image is taken and projected onto space spanned by the principal components. Additionally, a set of primary driving commands are measured. Finally, the position information is estimated and added as a 3-dimensional vector value. The motivation behind combining the driving parameters with auxiliary image data is that human driving behaviour relies significantly on visual information. The change in the image representation, resulting from changing the position is tightly correlated to the set of consecutive commands that are applied.

Finally, the dataset may be seen as a CSV table, illustrated in Figure 5.3, consisting of the vector values that correspond to the individual timestamps within the simulation. Each of the vector values consists of the following:

- Timestamp

T1	POSITION (T1)	COMMANDS (T1)	IMAGE PROJECTION (T1)
T2	POSITION (T2)	COMMANDS (T2)	IMAGE PROJECTION (T2)
T3	POSITION (T3)	COMMANDS (T3)	IMAGE PROJECTION (T3)
T4	POSITION (T4)	COMMANDS (T4)	IMAGE PROJECTION (T4)
T5	POSITION (T5)	COMMANDS (T5)	IMAGE PROJECTION (T5)

Figure 5.3: An illustration of the data organization. The rows correspond to the consecutive time samples in the Phabmacs simulation, while the columns correspond to the particular information type. Each entry represents a feature , measured at a specific time.

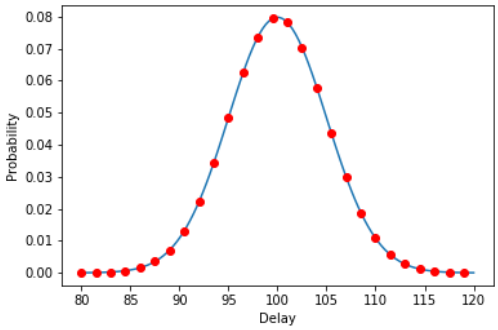


Figure 5.4: Example illustration of the delay probability distribution

- Camera image projected onto space spanned by the principal components
- Vehicle Position
- Vehicle primary commands that are currently applied

Training the neural networks is done using this CSV table. The input data is a sequence of the above-defined vectors. The target commands are the delayed commands that correspond to the expected network delays. Given that sudden braking and changes in the steering angle might occur with discrete nature, we designed a technique that handles this problem. Instead of providing a single target value during training, we multiply the delayed driving parameters by the probability of that delay occurring in the network component in the Tele-Operated system. Applying the technique, as mentioned above, benefited the overall driving smoothness and generalization performance.

This technique can be explained with the aid of the example Gaussian distribution, depicted in Figure 5.4. Let us assume that every point corresponds to the time stamp,

characterized by the set of driving commands. During the training phase, the neural network is provided with a sequence of input values. In addition, the desired target is also provided. Using the input-output relationship and the backpropagation algorithm computes the optimal set of parameters to minimize the error between network predictions and target values. To create the target command, each command in the training set is multiplied with the probability of the network delay corresponding to it. In other words, commands that are tight around the expected delay are multiplied with higher coefficients, while the commands that are received with unexpected delay are multiplied with lower coefficients. This technique enables the introduction of "priority" and "importance" between commands. Finally, we add all the "weighted" commands, thus creating a weighted average as a target command. The property of all probabilities that they add to one is a fundamental part of this technique. In this example, target commands resemble mostly the commands around the peak of the distribution.

5.3 Architectural Design

This section contains an overview of the system architecture as well as descriptions for all relevant components, whereas more detailed information may be found in 6. Apart from auxiliary components that serve to enable Tele-Operation and communication between the vehicle and the server, we introduce a neural-network-based prediction mechanism for generating command predictions.

5.3.1 Auxiliary Components

The communication between client and server is fundamentally the same as with the Positional Prediction Approach, that is introduced in chapter 4. To summarize, vehicle and server communicate over the network using custom-designed messages. The messages traverse over the network that inevitably introduces delays. The nature of the occurring delays is unspecified in the sense that we only estimate the distribution as Gaussian. Still, the exact values vary according to the Gaussian probability function with variable mean and standard deviation. Variable delays imply that message overtaking may take place. The network delay component and the message stack component are implemented in the same fashion as defined in 4.3. In this section, we the only fundamentally different component is the message, which is introduced in the following.

The message format on which this approach is fundamentally based is particularly specific and displayed in Figure 5.5. Apart from the necessary information about the sender, receiver, send-time and receive-time, it is characterized by the message data. More precisely, each message has placeholders for the following:

- Current Image - An image, captured by the on-board camera. The image resolution is 256x256, and there are three channels, corresponding to the RGB colour model.
- Current Position - A 3-Dimensional vector that corresponds to the vehicle position at which the image was taken.

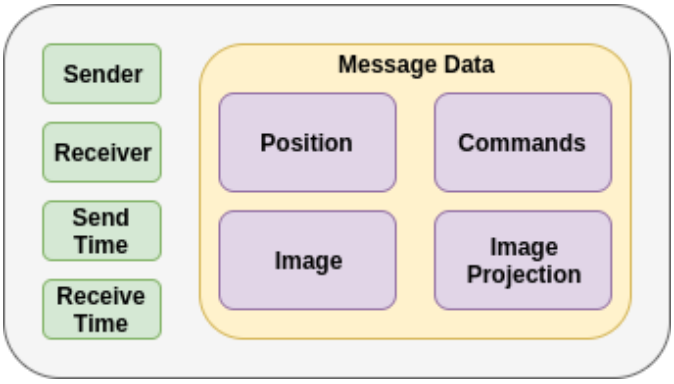


Figure 5.5: The illustration of the message format, that is used for the Command Predicting Approach in the Tele-Operation implementation

- Image projection - A vector of 44 values that correspond to projecting an image onto space spanned by the 44 most significant Principal Components.
- Steering Angle, Brake and Throttle parameters that correspond to the commands that

We used this message format both for inbound and outbound messages. A more carefully engineered approach would imply using two-types of messages, one from vehicle to the remote driver, and one in the other direction. We decided on this general approach to keep this proof-of-concept model lightweight and straightforward for future development.

Prediction Mechanism

For generating the command predictions, we rely on the set of previous commands, that reside on the message stack component. Furthermore, we rely on the set images that correspond to specific commands, as well as the location. The goal of a good predictor is to estimate commands concerning the correlation in the sequence of the previous information[4] that is available.

To design such command predictor, we utilize the concept of neural networks. In other words, we design neural networks that use the available information sequence and produce a set of commands. In comparison to the situation in which no command prediction mechanism is used, the newly predicted commands are more similar to the currently applied commands. Section 5.4 evaluates the performance of the proposed solution.

The proposed neural network has an input and the output interface. The input is a sequence of 30 most recent input vectors, corresponding to the 30 most recent messages. With this information at its disposal, the vehicle is responsible for generating future

[4]By the term information; we assume the command, corresponding image and location.

commands. Each input vector represents a concatenation of the PC-image-projection, corresponding position and corresponding commands. The outputs of the network are steering angle, brake and throttle[5].

The neural network developed for this purpose consists of four LSTM layers. They have 256,128, 64 and 32 neurons per layer, respectively. The final fully connected layer enables converging to the required output size. The activation function that is used is ReLU.

For training our neural network, we employ batch-gradient descent. In other words, we select a subset of 200 randomly chosen trajectories in the dataset and derive network inputs and target values. Upon computing the gradient that minimizes the error between model predictions and target values, we propagate the gradient through the network adjusting the network parameters. The before-mentioned process is commonly called backpropagation. We iterate this procedure 25000 times using the learning rate set as 0.001.

More information about the training phase is available in section 5.4.

5.4 Evaluation

Here we evaluate the given approach. After describing the evaluation methodology at the beginning of this section, we provide evaluation results and compare the scenario in which predictions are used for driving a vehicle with the scenarios in which human driving is employed.

5.4.1 Test Environment

To carry out the evaluation, the tests are performed partly in the Phabmacs simulation, partly in externally in Python during the training phase of the neural network. Scalability of the given approach is evaluated by iteratively sampling the simulation and externally saving the parameters necessary to conduct the comprehensive scalability analysis in Python afterwards. Section 5.4.2 provides a thorough examination. Usability evaluation, provided in 5.4.3, is conducted in a similar fashion. Performance evaluation is conducted during the training phase of the neural network training. This is done to evaluate how the training phase influences the overall solution. We can distinguish the phases in which the neural networks are outperformed by simply not using predictions. Similarly, we can identify a region in which the neural network approach is superior. This is thoroughly examined in section 5.4.4.

5.4.2 Scalability

The computational efforts of this approach are dependent solely on the complexity of the prediction model. They do not increase nor decrease with varying network delays.

[5]Brake and throttle are merged into one parameter(throttle-brake) because they cannot be simultaneously driven. Negative values correspond to brake values, while positive values correspond to throttle values.

Similarly, the increase or decrease in the number of surrounding vehicles does not influence the time required to carry out these computations. To argue that the proposed approach is sufficiently scalable, we measure the average time necessary to compute the output of the neural network. Similarly, we investigate the FPS trend in the simulation.

Average FPS	FPS variance
25.2784	1.2453

Table 5.1: The tabular overview of the simulation FPS when performing Command Prediction Approach.

When adopting the approach in which machine-learning models are directly responsible for driving the vehicle, the simulation maintains an average of around 25 frames per second. Furthermore, low variance implies that the performance oscillations and peaks occur rarely.

5.4.3 Usability

The variance of the consecutive commands may measure the usability of our proposed solution. Similar to the evaluating jitter in section 4.4.3, we measure how similar are consecutive estimations. The increase in the similarity between the consecutive commands implies "smooth" driving without sudden changes in the driving parameters. In contrast, high variance implies that driving consists of ofter-occurring sudden spikes in the throttle or brake parameter or sudden changes in the steering angle. Therefore, we argue that usability is inversely proportional to the prediction variance. By further decreasing this variance, we contribute to increasing usability.

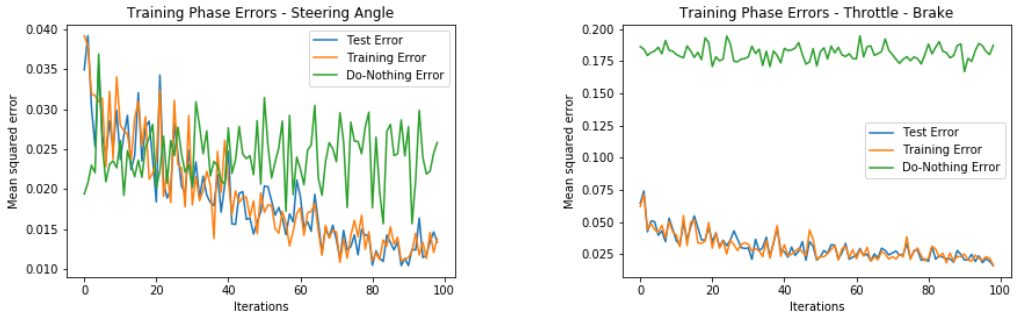
Approach	Average Jitter	Jitter variance
Without Prediction	0.0102	0.0022
With Command Prediction	0.0179	0.0022

Table 5.2: The tabular overview of the average usability parameter and its variance .

Table 5.2 describes the usability comparison between human driving and neural-network driving. This is simply a measure of driving "smoothness". Although the on average human driving achieves fewer oscillations between consecutive commands, and therefore, these systems are more usable, the machine-learning command prediction approach closely follows this performance.

5.4.4 Performance Measurements

The performance of the command-predicting neural network is closely followed during its training phase. The results are illustrated in Figure 5.6. The success of predicting the steering angle is particularly interesting and depicted in Figure 5.6a. We observe that



(a) Steering Angle parameter prediction mean error

(b) Thtottle-Brake parameter prediction mean error

Figure 5.6: Performance evaluation during training phase in terms of average error for a set of 200 trajectory samples.

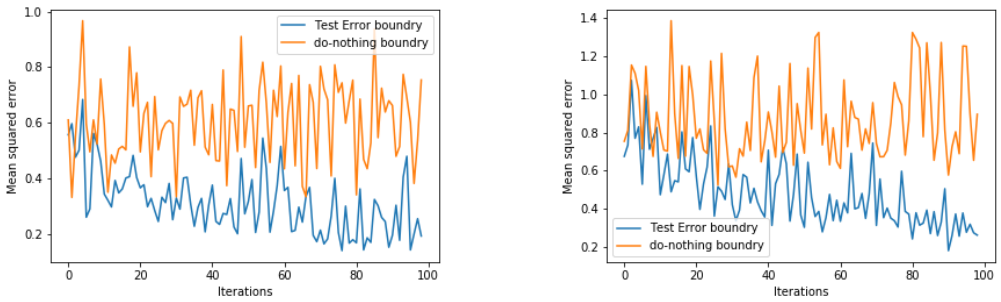
there are two regions. The first region, in which human driving without predictions[6] outperforms the neural-network-controlled driving[7] The second region is the region in which the neural networks are superior. Furthermore, as the training phase progresses, the difference between the two approaches increases. Figure 5.6b depicts the performance difference between the machine-learning generated throttle-brake parameter and the parameter that is received from the human driver. Since brake and throttle are not to be applied simultaneously, we have merged them into one parameter where negative values correspond to the braking parameter, and positive values are throttle. For example, a throttle-brake parameter of -0.3 corresponds to applying 30% of the maximum brake without any throttle. In contrast, a throttle-brake parameter of +0.2 corresponds to applying 20% of the throttle without braking.

Apart from the average error during the training phase, one of the first concerts is the tightness of error distribution. This is an absolute difference between the minimum and the maximum error achieved on a set of 200 sample trajectories. The results are depicted in Figure 5.7. We observe that the error boundary is tighter as the training progresses. Usually, this correlates positively to the error variance, and the lower error boundary implies that the errors tend to disperse less when performing machine-learning predictions, rather than relying on the delayed human commands. The results shown in Figures 5.7a and 5.7b instill confidence in this approach.

After achieving positive results during neural network training, the newly designed approach is deployed in the Phabmacs simulator. We conducted the performance evaluation by iteratively sampling the simulation and comparing the commands that are applied by the human driver on the server-side, commands that are applied by the neural network, and the commands that would have been applied in the absence of any prediction model. For the ease of comparison, we aim to compare one-dimensional er-

[6]Human driving is depicted with a green line

[7]The ML response on the training set is illustrated with the orange line, while the response to the test set is depicted using a blue line.



(a) Steering Angle parameter prediction error boundary

(b) Throttle-Brake parameter prediction boundary

Figure 5.7: A prediction performance during the training phase displayed as an error boundary within the set of 200 trajectory samples.

Approach	Average Error	Error variance
Human driving	0.0440	0.0120
Prediction-based driving	0.0569	0.0101

Table 5.3: The tabular overview of the performance between human driving and neural network-based driving.

rors, instead of three-dimensional ones[8]. We generate a one-dimensional array where each value is a square root of the sum of squares for the steering angle, throttle and brake errors. The performance measure between human Tele-Operated driving and its machine learning counterpart is described in Table 5.3. It is observable that the machine-learning approach for predicting the commands underperforms human driving without predictions. In contrast, the error variance of human driving is higher than the variance produced by the prediction model. We observe a certain disparity between the performance of the prediction model during neural network training and upon deployment in the Phabmacs simulator. This will be further addressed in section 7.2.

[8]Since the model predicts steering angle, throttle and brake, the error is three-dimensional.

6 Implementation

The architectural design, that was introduced in sections 5.3 and 4.3, relied upon a certain functional components. These components are described in this chapter in greater detail. In order to further elaborate our approach, as well as to ease the future work, built on top of this thesis, we organize this chapter in a concise and structured way.

Prior to providing more technical details about the implementation of specific components, we will present the working directory organization in order to have a more structured and concise view on the project. This will be done in section 6.1.

The simulations are done in the Phabmacs simulation environment. 6.2 describes the steps necessary to take in order to successfully start a simulation. Furthermore, section 6.2 introduces the basic variability and various scenario configurations.

Section 6.3 describes the implementation details related to operate a vehicle. Apart from the basic actuator setup, we will introduce a custom created controller mechanisms that are developed for this purpose.

In order to acquire information about its surroundings, the vehicle relies on various sensors. The information regarding the means of information gathering is presented in section 6.4. This section contains the methods and procedures that enable the vehicle to acquire positional knowledge about itself as well as other traffic participants. Furthermore, camera sensor and related functionality is described here.

The vehicle-to-vehicle communication is relied upon extensively in this work. Therefore, we describe the implementation of all communication mechanisms in the section6.5. Apart from the message format, stack organization and mechanisms, this section contains also the implementation for the network delay generation.

Finally, the implementation for the prediction modules is described in section 6.6. Although the main takeaway of this work are various ways of generating predictions, we leave the implementation details for the end of this chapter. Prior to the actual decision phase, the information needs to be captured and transmitted, and the mechanisms for applying the decision need to be implemented. Therefore sections 6.3, 6.4, 6.5 precede this section.

6.1 Code Organization

The source code for implementing Tele-Operated driving is written in Java and grouped into the following packages with respect to their functionality.

- **dcaiti.phabmacs.apps.studentStuff** The root of our development package, from which different subpackages branch depending on their functionality.

- **com.dcaiti.phabmacs.apps.studentStuff.Starters** contains the code for starting the given scenario. The Starters class in Java is responsible for initializing the given scenario, loading the map, moderating the traffic and other functionality, necessary for proper experiment setup. More details about the starting the scenario follows in section 6.2.

- **com.dcaiti.phabmacs.apps.studentStuff.utils** contains a code that implements auxiliary functionality. Among other, here is implemented the functionality that ensures storing simulation results into external CSV files. Moreover, streamers for motion JPEG images are implemented, as well as manager for generating network delays and XBoxController sampling mechanism.

- **com.dcaiti.phabmacs.apps.studentStuff.PCA** contains all the necessary java code that enables the Principal Component Analysis. Although PCA not a standalone component, but rather a auxiliare pre-processing step for the input into neural networks, PCA has significant importance and complexity. Therefore a separate package that implements PCA is developed.

- **com.dcaiti.phabmacs.apps.studentStuff.PPApproach** contains all the necessary classes and functions that implement a Position Prediction approach. An architectural diagram, that is built on top of majority of this package is presented in section 4.3

- **com.dcaiti.phabmacs.apps.studentStuff.CPApproach** contains all the necessary code that implements a Command Prediction approach. The architecture that further elaborates the high-level concepts, implemented in this package, are presented in section 5.3

6.2 Scenario Initialization

This section contains the necessary steps for proper experiment setup. The concept of a scenario is used in Phabmacs to encapsulate the experiment and one simulation run. Therefore, each experiment begins with the appropriate scenario initialization.

The primary step for starting the scenario is loading the adequate simulation map. Further, some traffic rules are then set up. For example, a scenario may be equipped with a maximum number of vehicles that are present or driving parameters that apply to all vehicles in the simulation. Only a subset of this information is displayed below.

```
HdMap map = new HdMap("maps/seestr_intersection.osm");
map.buildMap();
scenario = new Scenario(map);

scenario.setMaximumCarSpeed(50);
scenario.setMaximumNumberOfCars(50);
scenario.setMaximumCarAcceleration(5);
scenario.setMaximumCarDeceleration(3);
scenario.setPleasantCarAcceleration(2.5);
scenario.setPleasantCarDeceleration(2);
scenario.setMinimumSpawnDistance(10);
scenario.setRespawnCars(true);
scenario.setUseVisualizer(true);
```

Listing 6.1: Initializing scenario map

Finally, traffic participants are spammed to drive on the pre-computed routes, as the code snippet below suggests. In this example, a route is created as a set of consecutive points on the map. After the route configuration is added to the scenario, it is limited to a certain number of vehicles that use this route. Finally, the number of desired vehicles for this route is added, including the sensor and controller factories each vehicle is equipped with.

```
//NE -> SW
ArrayList<Long> routeB = new ArrayList<Long>();
routeB.add(311181586L);
routeB.add(27177983L);
RouteConfiguration route2 = scenario.addRoute(
"route2", routeB, -1, null, true);

scenario.setMaximumNumberOfCars(route2.getName(), carsPerRoute,true);
scenario.addVehicles(carModel, "route2", 15,
driveControllerFactories, sensorFactories);
```

Listing 6.2: Adding vehicles to follow pre computed routes

The Tele-Operation specific parts of scenario initialization are shown on the code snippet below. In other words, the two following methods are called depending on which scenario is desired. Delay manager is set up as an independent component in the simulation. Similarly, noise manager is also started. As the code below suggests, server and ego vehicle are added to the simulation. Finally, the connection between them is established.

```
1
2  public static void initializeCPApproach(
3      Scenario scenario, Simulator simulator) {
4
5      double delayMean = 0.2;       // assume 200ms delay
6      double delayVariance = 0.1;  // assume 10ms variance
7      DelayManager delayManager = new DelayManager(
8      delayMean,delayVariance);
9
10     double noiseMean = 0;          // assume non-biased system
11     double noiseVariance = 0.01;  // assume spread is 10cm
12     MeasurementNoiseManager noiseManager = new MeasurementNoiseManager(
13     noiseMean,noiseVariance);
14
15     // create server
16     CPServer server = new CPServer(scenario, delayManager, noiseManager);
17
18     // create client
19     CPEgoVehicle egoVehicle = new CPEgoVehicle(
20     scenario, simulator, "models/smart.properties",
21     delayManager, NoiseManager, server);
22
23     egoVehicle.setColor(0);
24
25     Vector3d egoVehicle_position = simulator.toVectorCoords(
26     new GeoLocation(52.5506584, 13.3520321), new Vector3d());
27
28     egoVehicle.setPosition(egoVehicle_position, 210);
29     scenario.addCar(egoVehicle);
30
31     // establish connection
32     server.connectToVehicle(egoVehicle);
33 }
```

Listing 6.3: Initializing Command Prediction Approach

In a similar fashion, the Command Prediction approach scenario is initialized. After creating measurement and delay manager, server and ego vehicle are created, and the connection between them is established.

```
public static void initializePPApproach(
Scenario scenario, Simulator simulator) {

    double delayMean = 0.2;       // assume 200ms delay
    double delayVariance = 0.05; // assume 50ms variance
    DelayManager delayManager = new DelayManager(delayMean,delayVariance);

    double noiseMean = 0.0;        // assume non-biased system
    double noiseVariance = 0.01;  // assume spread is 10cm
    MeasurementNoiseManager noiseManager = new MeasurementNoiseManager(
    noiseMean,noiseVariance);

    /* setup server*/
    PPServer server = new PPServer(scenario, delayManager, noiseManager);

    /* setup ego car */
    PPEgoVehicle egoCar = new PPEgoVehicle(
    scenario, simulator, "models/smart.properties",
    delayManager, noiseManager, server);

    egoCar.setColor(0);
    Vector3d egoCar_position = simulator.toVectorCoords(
    new GeoLocation(52.5506584, 13.3520321), new Vector3d());

    egoCar.setPosition(egoCar_position);
    scenario.addCar(egoCar);

    /* establish connection between client and server */
    server.connectToVehicle(egoCar);

    /* set tracked */
    scenario.getVisualizer().setTrackedVehicle(egoCar);
}
```

Listing 6.4: Initializing Position Prediction Approach

6.3 Actuators and Motion

The primary requirement for any automotive application is the implementation of the vehicle motion. Therefore, this section describes the necessary steps for driving the applying driving commands on a given vehicle in Phabmacs simulator. Furthermore, the implementation of the custom input controller is defined as the primary source of providing driver parameters in this application. For implementing the custom controller sampling, we relied on imports from LWJGL, a java library for gaming applications.

Before applying the commands, a user interface is used for providing necessary motion parameters. As described in section 3.1, we use an XBoxController equipped with a mini steering angle, as shown on Figure 3.1. Initially, a regular keyboard was used. The discrete nature of the input was a major constraint (the input was binary in a sense that the key is either pressed or not pressed). Therefore it did not correspond to the regular

driving (in which, for example, a throttle is pressed to some extent). A custom function is written for sampling the commands.

```
1
2     public void sampleController() {
3         controller.poll();
4
5         LS_X = controller.getAxisValue(1);
6         LS_Y = controller.getAxisValue(2);
7
8         RS_X = controller.getAxisValue(4);
9         RS_Y = controller.getAxisValue(5);
10
11        RT = controller.getAxisValue(6);
12        LT = controller.getAxisValue(3);
13
14    }
```

Listing 6.5: Function for sampling the controller input

The above function updates the relevant fields that correspond to the buttons[1] on the controller. Whenever the commands are needed, this function is invoked to update the button fields. This is usually done in an iterative fashion.

After sampling, the access to the specific controller button values is available by the adequate GET methods. Furthermore, we have hard-coded buttons and their semantic meaning in this way. The example below describes how the throttle is sampled. Since the controller output is in the range between -1 and 1, but Phabmacs simulator expects a throttling parameter in the range between 0 and 1, adjustments had to be made.

```
1 public double getThrottle() {
2
3     return (this.RT + 1) / 2 ;
4
5 }
```

Listing 6.6: Function for accessing the throttle parameter, which is mapped to the right trigger

Another interesting observation is that the available controller has a "dead zone" for both sticks. In other words, we have empirically investigated that values smaller than 0.05 are not detected, and the first value that is detected is 0.05. Such production imperfections are corrected in software, as the code below exemplifies.

[1]The first letter of the button nomenclature is an indicator of the side (L=left, R=Right). The second letter is the button type (T=Trigger, S=Stick). If the button has two axes of motion, they are indicated by X and Y

```
public double getSteeringAngle() {

    if(LS_X > 0.05) {
        return LS_X - 0.05;
    }
    if(LS_X < -0.05) {
        return LS_X + 0.05;
    }
    return LS_X;
}
```

Listing 6.7: Function for applying user commands to the vehicle

So far, we have presented a concise and structured way to transform the user input to the set of primary driving commands. The fundamental driving commands that we used in this thesis are throttle, brake and steering angle. Although the Phabmacs simulator is equipped with a broader range of possible driving actions, some of which are controlling blinkers or hand brake, we relied only on the beforementioned three. Since this work is a proof of concept, the three commands were sufficient. Setting driving parameters is straightforward and is shown on the code snippet below.

```

public void applyUserCommands(Vehicle v) {

    v.setSteerAngle(getSteeringAngle());
    v.setThrottle(getThrottle());
    v.setBrake(getBrake());

}
```

Listing 6.8: Function for applying user commands to the vehicle

6.4 Sensors

In general, Sensors are used for gathering information from the surrounding. While conducting the research on this topic, we have extensively used the input from the camera sensor. Furthermore, location information is also collected directly from the Phabmacs simulator. Although not implemented as a separate sensor for convenience, we use the GPS features that are available in Phabmacs. This section provides an insight into how the camera is created and used in Phabmacs, as well as how the vehicle gathers information about its surrounding objects and their locations.

The camera sensor is to be mounted on the vehicle and can collect the images of its environment. Therefore, there is a possibility to utilize both rearview and side views for automotive applications. Since this is a proof-of-concept work, only front camera was used. The following code implements the image processor. It is not purely our work but is rather built on top of the implementation for the similar functionality from a colleague from Fraunhofer. The following code is responsible for obtaining an image from Phabmacs visualizer as a screenshot of a current visualizer context. The image is

then post-processed to the adequate form for external storing or streaming.

```
1  package com.dcaiti.phabmacs.apps.studentStuff.PCA;
2
3  import java.awt.image.BufferedImage;
4  import java.nio.ByteBuffer;
5
6  import com.dcaiti.phabmacs.visualizer.PhabmacsGlContext;
7  import com.dcaiti.phabmacs.visualizer.Visualizer;
8  import com.dcaiti.phabmacs.visualizer.lightgl.LightGlContext;
9  import com.dcaiti.phabmacs.visualizer.lightgl.util.BufferHelper;
10
11 public class ImageProcessor {
12     // sensor that is used for collecting images
13     private PassiveCameraSensor camera;
14
15     // image related data
16     private final int imageWidth;
17     private final int imageHeight;
18     private final int[] imgData;
19
20     private final Visualizer visualizer;
21
22     public ImageProcessor(Visualizer visualizer,
23     PassiveCameraSensor egoCamera) {
24         this.camera = egoCamera;
25
26         this.imageHeight = egoCamera.getHight();
27         this.imageWidth = egoCamera.getHight();
28         this.imgData = new int[imageWidth * imageHeight * 3];
29
30         this.visualizer = visualizer;
31     }
32
33     public BufferedImage captureImage() {
34
35         // get current context
36         LightGlContext glContext =
37         this.visualizer.getGfxEngine().getContext();
38
39         // make a placeholder buffer
40         ByteBuffer buf = BufferHelper.createByteBuffer(
41         imageWidth * imageHeight * 4);
42
43         // store the screenshot in the placeholder buffer
44         visualizer.makeScreenshot(
45         glContext, imageWidth, imageHeight, buf);
46
47         // use camera images
48         if(this.camera != null) {
49             this.camera.makeScreenshot(
50             (PhabmacsGlContext)
51             this.visualizer.getGfxEngine().getContext());
52             buf = this.camera.getCurrentImage();
```

```
53          }
54
55          // create an image placeholder
56          BufferedImage img = new BufferedImage(imageWidth,
57          imageHeight, BufferedImage.TYPE_3BYTE_BGR);
58
59          // convert image
60          convertImage(buf, img);
61
62          return img;
63      }
64
65      /**
66       * Method for converting ByteBuffer data into BufferedImage data
67       * @param buf image in ByteBuffer format
68       * @param img image in BufferedImage format
69       */
70      private void convertImage(ByteBuffer buf, BufferedImage img) {
71          int lineStride = imageWidth * 3;
72          for (int y = imageHeight - 1; y > 0; y--) {
73              int start = lineStride * y - 4;
74              for (int i = start; i < start + lineStride; i += 3) {
75                  imgData[i+1] = (int) buf.get() & 0xff;
76                  imgData[i+2] = (int) buf.get() & 0xff;
77                  imgData[i] = (int) buf.get() & 0xff;
78                  buf.get();
79              }
80          }
81          buf.flip();
82          img.getRaster().setPixels(0, 0, imageWidth, imageHeight, imgData);
83      }
84  }
```

Listing 6.9: Class for creating camera images

While conducting this research, we have relied on the positional and directional information for all the vehicles. In Pabmacs, this information is available for each vehicle using the simple code below. In the form of a 3-dimensional vector, this information is available for further usage.

```
1 Vector3d position = new Vector3d(egoCar.getPosition(new Vector3d()));
2 Vector3d heading = new Vector3d(
3 egoCar.getDynamicsModel().getFrontDirection(new Vector3d()));
```

Listing 6.10: Obtaining information about vehicle direction and position

6.5 Communication

Communication is an essential part of any Tele-Operation application. It enables the sensory information to be transmitted to the server, as well as the commands to be sent to the vehicle. In this section, we describe the implementation of the message passing

concept that is developed.

The message between client and server is the core building block of the communication implementation. Primarily, each message contains a set of information that is to be shared between the two communicating parties. In our case, both inbound and outbound messages share the same format. The code snippet below shows the message data implementation class.

```
1  package com.dcaiti.phabmacs.apps.studentStuff.CPApproach;
2
3  import java.awt.image.BufferedImage;
4  import org.pmw.tinylog.Logger;
5  import com.dcaiti.phabmacs.sim.math.Vector3d;
6
7  public class CPMessageData {
8
9      private double   time;
10
11     private double   steeringAngle;
12     private double   throttleBrake;
13
14     private Vector3d position;
15
16     private BufferedImage currentImage;
17     private double[] imagePCAProjection;
18
19     public CPMessageData() {
20     }
21
22     public void fillDataAsEgoCar(double time,
23     Vector3d position, BufferedImage currentImage) {
24
25         this.time = time;
26         this.position = position;
27         this.currentImage = currentImage;
28     }
29
30     public void fillDataAsServer(double[] imagePCAProjection,
31     double steeringAngle, double throttleBrake) {
32
33         this.imagePCAProjection = imagePCAProjection;
34         this.steeringAngle = steeringAngle;
35         this.throttleBrake = throttleBrake;
36     }
37 }
```

Listing 6.11: Class implementation message data

This format enables expanding the information set by adding further details in the lightweight fashion. The only requirement is to add the necessary information to the set of class fields and to extend the function implementation, which populates this field accordingly. For the case of the command prediction approach, the ego vehicle sends its position with the current camera image. Upon reception, the server computes the prin-

cipal components of this image and returns it with a set of primary driving commands. As a result, after the roundtrip of a message, the vehicle can establish a connection between the camera images and the corresponding commands. This information is used when generating predictions.

```
package com.dcaiti.phabmacs.apps.studentStuff.CPApproach;

import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.List;
import org.pmw.tinylog.Logger;
import com.dcaiti.phabmacs.sim.math.Vector3d;

public class CPMessage {

    // message source and destination
    private CPEgoVehicle egoVehicle;
    private CPServer server;

    // timing parameters
    private double sendTime;
    private double receiveTime;

    // message data
    private CPMessageData messageData = new CPMessageData();

    public CPMessage(CPServer server, CPEgoVehicle egoVehicle,
    double sendTime, double receiveTime) {

        this.server = server;
        this.egoVehicle = egoVehicle;

        this.sendTime = sendTime;
        this.receiveTime = receiveTime;
    }

    public void fillMessageAsEgoCar(double time, Vector3d position,
    BufferedImage currentImage) {
        this.messageData.fillDataAsEgoCar(time, position, currentImage);
    }

    public void fillMessageAsServer(double[] imagePCAProjection,
    double steeringAngle, double throttleBrake) {
        this.messageData.fillDataAsServer(imagePCAProjection,
        steeringAngle, throttleBrake);
    }
}
```

Listing 6.12: Class implementation for the Command Prediction message

Its sender may characterize the message, recipient, send and reception time, as well as with the data, that is to be shared. Therefore, the implementation of the message class contains precisely this information.

Since the server and ego vehicle have different roles in communication, we distinguish two different approaches to how the message is to be filled with information. As discussed before, a vehicle populates the message with its position and image and sending time. At the same time, the server provides information about principal components of the image and the according commands.

Component for storing messages is called message stack. Both vehicle and server contain message stacks that have the purpose of storing most recently received messages. The message stack also serves as a source of information when production predictions. Apart from the storage functionality, the stack component is required to handle the out-of-order messages. In other words, as a result of different message paths, unpredictable delays and congestion in a network, it is not uncommon that messages arrive at their destination in order, that does not correspond to the order at which they are sent. Therefore, the stack component is required to rearrange the messages with respect to the time at which they are sent.

Message reception is described on the code listing below. When sending messages, they are scheduled to be received in the future. The stack component iterates over all scheduled messages. In this iterative fashion, it checks which messages are due to be received by comparing the current time and the message received time. If the message receive time is greater than the current time, that means that the message is to be received at some point in the future as it is not yet available. In contrast, if the current time is greater than the receive time, the message is received and added to the stack of received messages.

```
1  public void receiveMessages(double currentTime) {
2
3      List<CPMessage> msgsToRemove = new ArrayList<>();
4
5      // iterate over scheduled messages & add them to stack
6      for(int cnt = 0; cnt < this.scheduledMessages.size(); cnt ++) {
7
8          CPMessage tmp_message = this.scheduledMessages.get(cnt);
9
10         if(tmp_message.getReceiveTime() <= currentTime) {
11
12             addMessageToStack(tmp_message);
13             msgsToRemove.add(tmp_message);
14
15         }
16     }
17     // remove all the messages that are received
18     for(int counter = 0; counter < msgsToRemove.size(); counter ++) {
19         this.scheduledMessages.remove(msgsToRemove.get(counter));
20     }
21 }
```

Listing 6.13: Function implementation for receiving messages

The message reordering occurs in the function that adds the message to stack. First, if the message is already received, the function returns without storing. Afterwards, the

message is stored at the top of the stack. In an iterative process, the stack is arranged by the send time parameter by swapping the consecutive elements that are not received in the natural order.

```
private void addMessageToStack(CPMessage message) {

    int currentSize = receivedMessages.size();

    // check for duplicate messages
    // if duplicates found, don't add
    for(int counter = 0; counter < currentSize; counter ++) {
        if(message.getSendTime() ==
        this.receivedMessages.get(counter).getSendTime()){
            //"Duplicate message found
            return;
        }
    }

    // add message to the end of the stack
    this.receivedMessages.add(currentSize, message);

    // iterate over all message stack
    // find messages that are out-of-order
    // exchange out-of-order messages

    for(int counter = (currentSize); counter >= 1 ; counter--) {
        if(this.receivedMessages.get(counter).getSendTime() <
        this.receivedMessages.get(counter - 1).getSendTime()) {

            Collections.swap(receivedMessages, counter -1, counter);
        }
    }

    // remove oldest message
    if(this.receivedMessages.size() > this.stackSize) {
        this.receivedMessages.remove(0);
    }

    return;
}
```

Listing 6.14: Function implementation for adding messages to the message stack

Finally, the critical aspect when analyzing any network is its latency. Although unpredictable, network delays are always present in real-world networks. To make Phabmacs simulations as close to the real-world scenario as possible, we have designed a network delay manager, as the code listing below describes. Although network delays are often more complicated than random function with Gaussian distribution, for the proof-of-concept work, the implementation below was sufficient.

```
1  package com.dcaiti.phabmacs.apps.studentStuff.utils;
2
3  import java.util.Random;
4
5  public class DelayManager {
6
7      private double mean;
8      private double variance;
9      private Random rand = new Random();
10
11     public DelayManager(double mean, double variance) {
12         this.mean = mean;
13         this.variance = variance;
14     }
15
16     public double getRandomDelay() {
17         return this.variance * Math.abs(rand.nextGaussian())
18         + this.mean;
19     }
20
21     public void configureDelayManager
22         (double newMean, double newVariance) {
23
24         this.mean = newMean;
25         this.variance = newVariance;
26     }
27
28     public void setDelay(double newMean, double newVariance) {
29         this.mean = newMean;
30         this.variance = newVariance;
31     }
32 }
```

Listing 6.15: Class implementation for the network delay manager

6.6 Prediction Modules

This section provides in-depth implementation details. Initially, we will describe simpler prediction mechanisms that are used to predict vehicle positions. Further, we will describe the complete chain of collecting data, modelling training and exporting neural networks and finally, their usage in Phabmacs simulator.

While conducting the research on Tele-Operated driving, we have decided to estimate the future trajectories with both simple and complex prediction mechanisms. Therefore, the underlying execution platform does not have to have significant computational performance, which is often the case in embedded applications. In such cases, more straightforward approaches as regression models provide adequate performance. On the other hand, in case the available platform is equipped with enough computing power, it can be adequately utilized with more sophisticated approaches, such as neural networks.

The colleagues at the Fraunhofer Fokus implemented the deterministic trajectory prediction. In their implementation, the future trajectory is computed using the information about the vehicle pose, local longitudinal and lateral acceleration, as well as the current velocity. The code resides in the class: OffsetCompensator.java in the package: com.dcaiti.phabmacs.apps.studentStuff.PPApproach .

```
public static List<VehiclePose> computeFurtureTrajectory(
    TOVBoxDynamics boxDynamics,
    double futureTime, double stepTime) {

    Vector3d angularVelocity = new Vector3d(
        boxDynamics.angularVelocity.x, boxDynamics.angularVelocity.y,
        boxDynamics.angularVelocity.z);

    Vector3d localAcceleration = new Vector3d(
        boxDynamics.localAcceleration.x, boxDynamics.localAcceleration.y,
        boxDynamics.localAcceleration.z);

    Vector3d frontDirection = new Vector3d(
        boxDynamics.frontDirection.x, boxDynamics.frontDirection.y,
        boxDynamics.frontDirection.z);

    Vector3d localVelocity = new Vector3d(
        boxDynamics.localVelocity.x, boxDynamics.localVelocity.y,
        boxDynamics.localVelocity.z);

    Vector3d position = new Vector3d(
        boxDynamics.position.x, boxDynamics.position.y,
        boxDynamics.position.z);

    Vector3d tmpVec = new Vector3d();

    double vz = -localVelocity.z;
    double az = MathUtils.clamp(-localAcceleration.z, -2, 2);
    angularVelocity.multiply(stepTime);

    List<VehiclePose> trajectory = new ArrayList<>();
    trajectory.add(new VehiclePose(position, frontDirection));

    for (double t = 0.0; t <= futureTime; t += stepTime) {
        frontDirection.rotate(angularVelocity.x, 1, 0, 0);
        frontDirection.rotate(angularVelocity.y, 0, 1, 0);
        frontDirection.rotate(angularVelocity.z, 0, 0, 1);

        double vLast = vz;
        vz += az * stepTime;
        if (Math.signum(vLast) != Math.signum(vz)) {
            break;
        }

        frontDirection.multiply(vz * stepTime, tmpVec);
        position.add(tmpVec);
        trajectory.add(new VehiclePose(position, frontDirection));
    }

    return trajectory;
}
```

Listing 6.16: Deterministic function for computing the future trajectory

As the code snippet above suggests, one of the most glaring shortcomings of deterministic trajectory prediction is the fact that it relies only on a single point in time. This means that high variance in measurement leads to high variation in predictions. Furthermore, if messages are received out-of-order, the predictions will be similarly out of order, in addition to the significant variance that is incurred by the measurement noise.

In order to prevent the scenario as mentioned above, we have decided to use the concept of polynomial and linear regressions in order to use multiple points in the prediction. First, we introduce simple linear regression, regularized by a constant of value 1. In order to predict a three-dimensional vector, each dimension is predicted independently. As a result, when performed on the target hardware, the computation may be parallelized across multiple threads or even multiple processing units.

```
1
2  public static Vector3d LinearRegressionPrediction(
3      double currentTime, List<Double> previousTimeStamps,
4      List<Vector3d> previousPositions)
5      throws Exception {
6
7      int size = previousTimeStamps.size();
8
9      /* create arrays from list data */
10     double[] time = new double[size];
11     double[] x = new double[size];
12     double[] y = new double[size];
13     double[] z = new double[size];
14
15     for(int i = 0; i < size; i++) {
16         time[i] = previousTimeStamps.get(i);
17         x[i] = previousPositions.get(i).x;
18         y[i] = previousPositions.get(i).y;
19         z[i] = previousPositions.get(i).z;
20     }
21
22     /* predict new positions */
23     double x_new = oneDimensionRidgeRegression(time,x,1)
24         * currentTime;
25
26     double y_new = oneDimensionRidgeRegression(time,y,1)
27         * currentTime;
28
29     double z_new = oneDimensionRidgeRegression(time,z,1)
30         * currentTime;
31
32     /* return prediction */
33     return new Vector3d(x_new,y_new,z_new);
34 }
```

Listing 6.17: Implementation of the Ridge regression

```
1
2  public static Vector3d polynomicRegressionPrediction(
3      double currentTime, List<Double> previousTimeStamps,
4      List<Vector3d> previousPositions, int degree ) {
5
6      WeightedObservedPoints x = new WeightedObservedPoints();
7      WeightedObservedPoints y = new WeightedObservedPoints();
8      WeightedObservedPoints z = new WeightedObservedPoints();
9
10     for(int i = 0; i < previousTimeStamps.size(); i++) {
11         double time = previousTimeStamps.get(i);
12         double x_current = previousPositions.get(i).x;
13         double y_current = previousPositions.get(i).y;
14         double z_current = previousPositions.get(i).z;
15
16         x.add(i^3, time, x_current);
17         y.add(i^3, time, y_current);
18         z.add(i^3, time, z_current);
19     }
20
21     PolynomialCurveFitter fitter=PolynomialCurveFitter.create(degree);
22
23     double x_prediction = fitAndPredict(fitter, x, currentTime);
24     double y_prediction = fitAndPredict(fitter, y, currentTime);
25     double z_prediction = fitAndPredict(fitter, z, currentTime);
26
27     return new Vector3d(x_prediction,y_prediction,z_prediction);
28 }
```

Listing 6.18: Implementation of the Polynomial regression

As the code above suggests, we have decided on using weights for the observed points in each dimension. Weighted observations enable favouring the newly received trajectory information. Further, function fitAndPredict is used to predict the trajectory in all three dimensions. An implementation for the fitAndPredict function is shown below.

```
1  private static double fitAndPredict(
2      PolynomialCurveFitter fitter, WeightedObservedPoints data,
3      double currentTime){
4
5      double[] coeff = fitter.fit(data.toList());
6
7      double prediction = 0;
8      for(int i = 0; i < coeff.length; i++) {
9          prediction += coeff[i] * Math.pow(currentTime, i);
10     }
11
12     return prediction;
13 }
```

Listing 6.19: Implementation of the fitAndPredict helper function

Since the prototyping and optimizing neural networks required significant efforts in comparison to developing other prediction mechanisms, we devote the remainder of this

chapter to the processes that relate to creating, optimizing and deploying LSTM neural networks. Before being able to use the neural networks, they first have to be designed and trained using the collected data. Finally, after achieving desired performance, the neural networks are exported into a suitable format, which enables their usage in the Phabmacs simulator.

Unlike traditional explicit programming in which the behaviour is a result of program execution, implicit programming techniques like machine learning rely on the data as a source of information. From data, this information is extracted and embedded into the final models using network optimization algorithms such as backpropagation. The quality of the overall model is proportional to the quality of the underlying data. Therefore the initial step in creating neural networks is collecting the data. This data is later used during the neural network training. Data Collection is a vital part of any machine learning pipeline. Therefore, we have designed a particular scenario in which a vehicle is driven by a human driver. During this test drive through Phabmacs simulations, the data is collected in the form of CSV files. Each row corresponds to the particular time step in the simulation, while each column represents one specific feature that is measured[2]. The implementation is contained in the PPADataCollector.java class. Here, we present only the partial but most significant scenario setup. The following code describes how we used methods for adding hooks to the visualization. In each simulation step, the controller is sampled, and the commands are applied to the vehicle. Using this rendering hook, we enable the periodic execution of our desired functionality.

```
1  private void setupDriving() {
2
3      // command sampling and applying
4      XboxController controller = new XboxController();
5
6      visualizer.getGfxEngine().addEngineListener(new GfxEngineAdapter() {
7          @Override
8          public void onRenderFrameFinished(LightGlContext glContext) {
9              // sample controller
10             controller.sampleController();
11             controller.applyUserCommands(egoCar);
12         }
13     });
14 }
```

Listing 6.20: Setup for the driving mechanism during data collection phase

The following code enables the essential functionality of any data collection mechanism. We have designed a simulation listener that, on each simulation step, computes the position and direction of the ego vehicle. In an iterative fashion, this information is exported to external CSV files, from which are further used during neural network training.

[2]For example, time, position or direction.

Figure 6.1: TensorFlow logo

```
private void startSampling() {
    visualizer.getGfxEngine().addEngineListener(
    new GfxEngineAdapter() {

    @Override
    public void onRenderFrameFinished(LightGlContext glContext) {
        double time = sim.getSimulationTime();

        Vector3d position = new Vector3d(egoCar.getPosition(
            new Vector3d()));

        Vector3d heading = new Vector3d(
            egoCar.getDynamicsModel().getFrontDirection(
            new Vector3d()));

        String row = createRow(time,position, heading);
        CSVFileWritter.writeRowToFile(row, outputFileName);
    }
    });
}
```

Listing 6.21: Implementation of the iterative sampling render hook

The main advantage of the python programming language is its extensive range of available machine learning libraries as well as the speed suitable for data pre-processing and model development. During project planning, we have decided to prototype, develop and optimize neural networks in Python using TensorFlow, Figure 6.1, before deploying them in Phabmacs. The primary reason behind this decision is the aforementioned extensive libraries that aid in machine learning tasks.

```
def createNetwork(networkInput, valueName):

    l1 = tf.contrib.rnn.BasicLSTMCell(activation=activation,
        num_units=LSTM_L1_units)
    l2 = tf.contrib.rnn.BasicLSTMCell(activation=activation,
        num_units=LSTM_L2_units)
    l3 = tf.contrib.rnn.BasicLSTMCell(activation=activation,
        num_units=LSTM_L3_units)

    lstm_layers = [l1, l2, l3]
    lstm_net = tf.contrib.rnn.MultiRNNCell(cells=lstm_layers)

    lstm_wrap = tf.contrib.rnn.OutputProjectionWrapper(
        lstm_net, output_size=1)

    # Prediction function and forward propagation
    LSTM_prediction, states = tf.nn.dynamic_rnn(
        lstm_wrap, networkInput, dtype=tf.float64)

    networkOutput = tf.contrib.layers.fully_connected(
        tf.layers.flatten(LSTM_prediction),
        num_outputs = 3,
        activation_fn=tf.keras.activations.linear)

    return networkOutput
```

Listing 6.22: Function for creating a neural network, comprised of stacked LSTM layers

Upon achieving satisfying results during training and evaluation, the neural network is exported into the format that is suitable for importing into Java applications, such as Phabmacs simulator. Using the term exporting in this context usually implies creating some sort of representation for the according to the computational graph. The computational graphs are simply a relatively large set of operations[3] connected in a directed network in which the output of source node is the input to the sink node. This graph of operations is, in essence, the neural network and the parameters are optimized during training.

```
        builder = tf.saved_model.builder.SavedModelBuilder(network_path)
        builder.add_meta_graph_and_variables(sess,
            [tf.saved_model.tag_constants.SERVING])
        builder.save()
```

Listing 6.23: Exporting Neural netowrks models

The final step in using the created machine learning models is deploying them directly in Phabmacs. For this purpose, they are converted in the adequate computational graphs, suitable for importing.

Using neural networks in Phabmacs simulator requires the following imports. These imports enable the design environment to import networks, provide adequate inputs and

[3] Usual operations in Neural Networks are multiplications, additions and nonlinear transformation using exponential functions.

collects outputs.

```
import org.tensorflow.SavedModelBundle;
import org.tensorflow.Session;
import org.tensorflow.Tensor;
```

Listing 6.24: Necessary imports for working with neural network graphs in java

The initial step for working with computational graphs is successful import into the Phabmacs simulator. The following code serves this purpose.

```
public static SavedModelBundle importModel(String modelDirectory) {
    try {
        return SavedModelBundle.load(modelDirectory, "serve");
    }catch(Exception e){
        Logger.error(e);
        return null;
    }
}
```

Listing 6.25: Code snippet for importing neural network graphs

After successful import, neural networks are ready to be provided with the adequate inputs, for which they produce an output. The following code demonstrates how to start a TensorFlow session, feed the network with the input tensor and collect the output values.

```
public double[] computeNetworkOutput(double[] networkInput) {

    try{
        // create a session
        Session sess = this.neuralNetworkModel.session();

        DoubleBuffer db = DoubleBuffer.wrap(networkInput);
        long[] networkInputShape = {
            1, this.numTimeSteps, this.networkInputSize};

        // create tensors from input data
        Tensor<Double> inputTensor = Tensor.create(networkInputShape,db);

        // get ML output
        Tensor<Double> result = sess.runner()
                .feed(this.networkInputNode, inputTensor)
                .fetch(this.outputNode).run().get(0).expect(Double.class);

        // reshape output
        DoubleBuffer a = DoubleBuffer.allocate((int)this.outputShape[1]);
        result.writeTo(a);
        double[] ret_val = a.array();

        return ret_val;
    } catch(Exception e) {
        Logger.error(e);
    }
    // if this is reached send commands not to move
    double[] ret_val = {-1.0,-1.0,-1.0};

    return ret_val;
    }
}
```

Listing 6.26: Computing the output of a neural network

7 Conclusion

The final chapter summarizes the thesis. The first subsection outlines the main ideas behind our conceptual implementation and recapitulates the work steps. Further, the issues that remained unsolved are described. Finally, the potential of the proposed solution and future work is surveyed in an outlook.

7.1 Summary

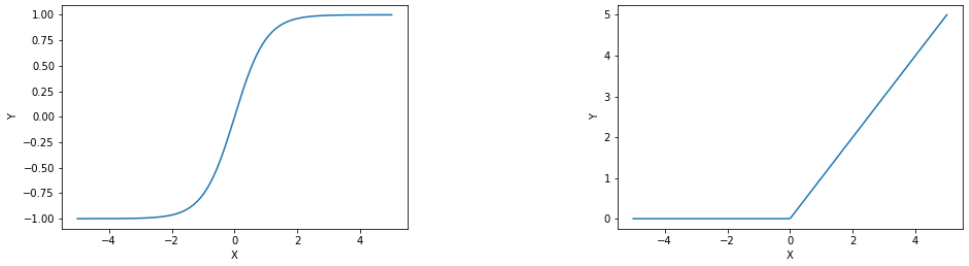
This thesis tackled the latency related issues for Tele-Operated driving. The network is responsible for the communication between the remote driver and the vehicle. Inevitably, the network introduces latencies. Resulting, the driver observes an outdated scene. Similarly, the vehicle applies outdated commands. To reduce the adverse effects of the network delays, we propose particular techniques, applicable both on the vehicle-side, as well as the server-side of the system for mitigating the consequences of network delays.

To tackle network delays on the server-side, we propose a range of trajectory prediction mechanisms that range from deterministic velocity-based to machine-learning-based, such as regression and neural networks. Future positions of all objects that participate in the traffic may be accurately estimated using these prediction mechanisms. Ultimately, this leads to effectively eliminating the network delays that cause the information on the server-side to be outdated. Our results demonstrate that deterministic approaches scale better with respect to the low computational efforts that they require. Furthermore, these approaches showed the most confident results in terms of lowest error. In terms of usability, the regression methods show the best performance.

Similarly, we propose a prediction mechanism based on recurrent neural networks for the computation of the unreceived commands. These neural networks rely on the sequence of previous commands, appended with auxiliary image data to estimate commands that closely resemble those, currently applied by the remote driver. The evaluation results show the potential to employ artificial intelligence to the concept of Tele-Operated driving. Prior to deployment, our custom-designed neural networks outperform human driving. After deployment, the results indicate similar performance.

7.2 Problems Encountered

Throughout this thesis, we have encountered most problems that relate to neural networks. Coming across these problems was expected, as this was the main field of experimenting in this topic.



(a) Hyperbolic tangent activation function

(b) ReLU activation function

Figure 7.1: Illustration of the two activation functions that were used in this project.

Primary, the problem that arose was how to design and run neural networks in Phabmacs simulator. Motivated by the previous success in applying machine learning to the automotive applications, we decided to conduct the design, train and evaluation phases externally using Python and Tensorflow. Afterwards, created models were exported and finally imported into Phabmacs simulations using particular Java libraries.

During the training and evaluation of the data, the main concern is often the lack of data for designing optimal models. We have overcome this issue by designing independent simulations with the purpose of pure data collection. Therefore, collecting data corresponds to "driving a car in the simulation". The resulting data is exported into CSV tables, suitable for merging or independent processing. The benefit of this approach is to design a data collection mechanism that is easily scalable and parallelizable for further work.

Initially, controlling the vehicles in Phabmacs simulator was done using a keyboard as input. Unfortunately, throttle, brake and steering angle have continuous nature, rather than discrete nature. Simple pressing the keyboard buttons did not meet the demands of our system. Therefore, we have acquired an Xbox Controller, equipped with the custom-designed wheel extension. Using this input, we were able to achieve a more realistic performance.

During the neural network training, we have encountered the problem, commonly known as a problem of vanishing gradient. Namely, training optimizations require computing the first derivative of activation functions. Initially, we used the hyperbolic tangent activation function because it corresponded to the span of the commands that we aimed to predict. The issue with hyperbolic tangent is that when the input is in the "flat" region, the derivative is practically close to zero, and therefore while learning, no significant optimization is performed. Instead, we used ReLU activation function, that does not exhibit this behaviour. Figure 7.1 depicts both activation functions.

During the training of neural networks, there is a significant difference between the accuracy when predicting the steering angle and the brake and throttle parameters. Furthermore, the neural network finds it challenging to outperform human driving when producing the steering angle. We argue that driving often consists of straightforward

trajectories, and therefore, our custom-designed dataset lacks information about steering manoeuvres. This information would enable the neural networks the possibility to generalize. In the future work on this topic, we will try to create a more comprehensive dataset that consists of longer driving time.

When predicting the unreceived commands, we observe the performance disparity between neural network training phase and the run-time performance upon deployment. The results in training prove are reassuring and produce confidence that this approach easily outperforms human driving. In contrast, the performance during run-time simulations shows that it is not easy to outperform human driving in real life. We suspect that it is due to the relatively small dataset that improperly encapsulates actual driving. One approach in overcoming these issues is to create a significantly larger dataset.

While conducting our research, the initial idea was to combine CNNs and LSTM neural networks, similar to what is presented in [19]. A research group from Stanford proposed combining temporal and spatial information of an image stream. Using this information, this research group proposes a prediction of steering angles, similar to what is proposed in our work. A stack of consecutive 3D convolutional layers performs feature extraction from a base image of size 640x480. Additionally, residual connections are added to enable better gradient flow and better training, as suggested in [35]. Finally, two blocks of LSTM layers are added to correlate features from consecutive images. Although this approach seemed very interesting, we could not replicate this proposal because of data constraints. Firstly, the authors used a set of around 100000 frames which was not plausible in our case. Since we were responsible for data generation and had no prior data from Phabmacs simulations, collecting such a significant amount of data was outside the scope of our work. Furthermore, the data used in [19] is highly variable as it contains real-world scenarios which cover a wide variety of manoeuvres, weather and traffic situations. The neural network that is proposed in [19] contains 27 million connections and 250 thousand parameters, and we estimated that training it with our relatively small dataset would be infeasible.

The Phabmacs simulator has limited rendering capabilities, as can be seen in Figure 2.2a. As hinted in section 5.1, reduced colourspace implies two dominant colours, grey for buildings and purple for roads. Furthermore, the colourspace of the road is consistent, which means that there are not gradual differences between different parts of roads and buildings displayed in Phabmacs simulator. From here, we may draw the following conclusions:

- The reduced colourspace may have lead to the overfitting that we experienced. In case data was more variable in the sense of colours, the model may have had better generalization performance, instead of overfitting on only two colours. In the case where an increased number of principal components were used, we suspect that overfitting occurred partially because of reduced colour spaces.
- We were able to represent the images using a relatively small number of principal components. If the colour variations were stronger, as this is the case with real data, there might arise the necessity to use more principle components to retain

the required variance. Unfortunately, there is also a possibility that an increase in colour variability may lead to infeasible PCA representations, in contrast to Olivetti[33] example.

We hope that further work on this thematic will include using real world-images and provide performance comparisons between PCA embeddings of Phabmacs and real-world images.

7.3 Dissemination

For successful deployment of Tele-Operated driving systems for commercial use, the negative influences of network latencies have to be appropriately handled. The network that connects the vehicle with the remote driver inevitably introduces delays. The vehicle positions that are available on the server-side are outdated in the sense that the actual positions have not yet reached the server. Similarly, the commands that are available on the vehicle-side are outdated. This is because the current commands that are sent are received after the time interval. This interval corresponds to the time, which is needed for a message to traverse the network. The main contribution of this thesis is to propose approaches to circumventing the effects of network delays. This can be achieved on both server-side and vehicle side, and therefore, we consider both situations.

To bypass the scenario in which the information, available to the remote driver, is outdated, we propose applying prediction mechanisms on the server-side. The purpose of these mechanisms is to estimate the current positions of the vehicles. This is achieved by relying on the available information that is received in the past. Diverse prediction mechanisms range from deterministic trajectory computation to data-driven machine learning models such as linear and polynomial regression and neural networks. Using this information, the driver may be presented with the estimated positions of the vehicles. These estimations may span into the future that corresponds to the message delay. Therefore the driver can observe the probable vehicle positions. These positions correspond to the time at which the commands are going to be applied. Further details about the implementation and evaluation are available in Chapter 4.

Another point of tackling the network-related issues is the vehicle itself. As mentioned above, the vehicle receives an outdated set of commands as a direct result of network latency. We propose using LSTM neural networks to predict the unseen commands using the set of received commands. To make a more comprehensive model, we use images as an auxiliary source of information to compute commands predictions. The motivation behind utilizing image information is that it is most relied upon when driving. LSTM neural networks exploit the correlation between change in position, information and commands. Further details about the implementation and evaluation are available in Chapter 5.

The two approaches as mentioned above for tackling the network delays are conceptually different. One is performed on the server-side; the other is performed on the vehicle side. Furthermore, message contents are different since different information is sent and

received. Also, the client and server operate differently for both approaches. Finally, the approaches are evaluated by different metrics. Both approaches are not mutually exclusive and therefore implemented separately. They are independent of one another, and therefore the commercial application may rely on one or both approaches, and the variations on one approach do not affect the other approach.

7.4 Outlook

Although this work aimed to be a comprehensive implementation of the approaches that successfully tackle problems related to Tele-Operated driving, the amount of work surpassed the group size. Any of the core points in this thesis may be further improved. Given that this is a relatively novel field of science, we intend to consider it as one of the cornerstones for the future work of our colleagues.

Data collection is one of the main optimization points. Since we opted for some implicit-learning approaches, the quality of data collected has a direct positive influence on the overall solution. Identifying new trends, as well as more detailed data processing, may be useful for further improvements.

Furthermore, designing better architectures often has a positive contribution. Some changes in the hyperparameter choice, such as the number of layers in the network, learning rate, number of neurons per layer seems like a good way to optimize our approach. Therefore, an aspect of future work may tackle the problem from an engineering perspective.

Finally, to further improve the usage of image data in prediction mechanisms, we may imply convolutional neural networks for this purpose. The property of spatial correlation is the main property of these networks, and they are a natural fit for the problems of this type.

List of Acronyms

PCA	Principal Component Analysis
LiDAR	Light Detection and Ranging
DNN	Deep Neural Network
WSN	Wireless Sensor Network
LSTM	Long short-term memory
RNN	Recurrent Neural Network
RGB	Red-Green-Blue colour image
CSV	Comma Separated Values
PHABMACS	PHysics Aware Behavior Modeling Advanced Car Simulator
ADAS	Advanced Driver Assistance System
RMS	Root-mean-square
C-LSTM	Convolutional Long Short-Term Memory
FPS	Frames Per Second
LWJGL	Lightweight Java Game Library

Bibliography

[1] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," *CoRR*, vol. abs/1708.08559, 2017.

[2] M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner. 05 2016.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] I. T. Forum, "Automated and autonomous driving," no. 7, 2015.

[5] T. Fong and C. Thorpe, "Vehicle teleoperation interfaces," *Auton. Robots*, vol. 11, pp. 9–18, July 2001.

[6] T. Tang, F. Chucholowski, and M. Lienkamp, "Teleoperated driving basics and system design," *ATZ worldwide*, vol. 116, pp. 16–19, 02 2014.

[7] H. Alshamsi and V. KÃ≪puska, "Smart car parking system," *International Journal of Science and Technology*, vol. 5, pp. 390 – 395, 08 2016.

[8] A. Khanna and R. Anand, "Iot based smart parking system," in *2016 International Conference on Internet of Things and Applications (IOTA)*, pp. 266–270, Jan 2016.

[9] K. Hassoune, W. Dachry, F. Moutaouakkil, and H. Medromi, "Smart parking systems: A survey," in *2016 11th International Conference on Intelligent Systems: Theories and Application (SITA)*, pp. 1–6, Oct 2016.

[10] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A survey on low latency towards 5g: Ran, core network and caching solutions," *IEEE Communications Surveys Tutorials*, vol. 20, pp. 3098–3130, Fourthquarter 2018.

[11] A. Snow, U. Varshney, and A. Malloy, "Reliability and survivability of wireless and mobile networks," *Computer*, vol. 33, pp. 49 – 55, 08 2000.

[12] Li Ma, Zihuai Lin, Zijie Zhang, G. Mao, and B. Vucetic, "Improving reliability in lossy wireless networks using network coding," in *2013 IEEE International Conference on Communications Workshops (ICC)*, pp. 312–316, June 2013.

[13] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.

[14] R. Sun, *The Cambridge Handbook of Computational Psychology*. New York, NY, USA: Cambridge University Press, 1 ed., 2008.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[16] A. Graves, S. Fernández, and J. Schmidhuber, "Multi-dimensional recurrent neural networks," *CoRR*, vol. abs/0705.2011, 2007.

[17] M. A. Nielsen, "Neural networks and deep learning," 2018.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] A. Simpson, "Self-driving car steering angle prediction based on image recognition," 2017.

[20] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction," *CoRR*, vol. abs/1801.07962, 2018.

[21] B. Coifman and L. Li, "A critical evaluation of the Next Generation Simulation (NGSIM) vehicle trajectory dataset," *Transportation Research Part B: Methodological*, vol. 105, no. C, pp. 362–377, 2017.

[22] E. P. S. Y. Ahmad El Sallab, Mohammed Abdou, "Deep reinforcement learning framework for autonomous driving," 2017.

[23] M. Vitelli and A. Nayebi, "Carma : A deep reinforcement learning approach to autonomous driving," 2016.

[24] H. M. Eraqi, M. N. Moustafa, and J. Honer, "End-to-end deep learning for steering autonomous vehicles considering temporal dependencies," *CoRR*, vol. abs/1710.03804, 2017.

[25] A. Simpson, "Self-driving car steering angle prediction based on image recognition," 2017.

[26] S. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi, "Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture," *CoRR*, vol. abs/1802.06338, 2018.

[27] B. Kim, C. M. Kang, S. Lee, H. Chae, J. Kim, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," *CoRR*, vol. abs/1704.07049, 2017.

[28] A. Khosroshahi, E. Ohn-Bar, and M. M. Trivedi, "Surround vehicles trajectory analysis with recurrent neural networks," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2267–2272, Nov 2016.

[29] S. Neumeier, N. Gay, C. Dannheim, and C. Facchi, "On the way to autonomous vehicles teleoperated driving," in *AmE 2018 - Automotive meets Electronics; 9th GMM-Symposium*, pp. 1–6, March 2018.

[30] T. L. T. Chen, *Methods for improving the control of teleoperated vehicles*. PhD thesis, Universität München, 2015.

[31] F. Chucholowski, S. Büchner, J. Reicheneder, and M. Lienkamp, "Prediction methods for teleoperated road vehicles," in *CoFAT*, (München), 2013.

[32] S. Neumeier, E. A. Walelgne, V. Bajpai, J. Ott, and C. Facchi, "Measuring the feasibility of teleoperated driving in mobile networks," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*, pp. 113–120, June 2019.

[33] F. S. Samaria and A. C. Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of 1994 IEEE Workshop on Applications of Computer Vision*, pp. 138–142, Dec 1994.

[34] R. Espirito Santo, "Principal component analysis applied to digital image compression," *Einstein (SÃ£o Paulo, Brazil)*, vol. 10, pp. 135–9, 06 2012.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

Annex