TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Elektrotechnik und Informationstechnik
Professur für Methoden der Signalverarbeitung

Hybrid Machine Learning Methods for Vehicle Safety Applications

Amit Tulsidas Chaulwar

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Sebastian Steinhorst
Prüfer der Dissertation:
1. Prof. Dr.-Ing. Wolfgang Utschick
2. Prof. Dr.-Ing. Michael Botsch

Die Dissertation wurde am 28.09.2020 bei der Technische Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 26.05.2021 angenommen.

Table of Contents

Zusammenfassung

Die Trajektorienplanung ist eine zentrale Aufgabe für autonome Mobilitätsanwendungen. Eine besondere Herausforderung ist die sichere Trajektorienplanung für Fahrzeuge in kritischen Verkehrsszenarien mit mehreren statischen und dynamischen Objekten. In einem kritischen Verkehrsszenario sollte eine sichere Trajektorie so geplant werden, dass eine Kollision vermieden und falls dies nicht mehr möglich ist, die Folgen gemildert werden. Für solche Aufgaben werden traditionell modellbasierte Algorithmen genutzt, weil ein genaues Verständnis der Modelle, die Aktoren in sicherheitskritischen Anwendungen steuern, von entscheidender Bedeutung ist. Für komplexe Planungsaufgaben sind modellbasierte Algorithmen allerdings sehr rechenintensiv und können nicht im Fahrzeug eingesetzt werden, weil bordeigenen Rechenressourcen begrenzt sind.

Die meisten Algorithmen des maschinellen Lernens sind, obwohl sie in der Regel im Vergleich zu modellbasierten Ansätzen sehr effizient in Bezug auf die Rechenressourcen sind, nicht interpretierbar, da es sich um rein datenbasierte Methoden handelt. Hybride Algorithmen, eine Kombination von maschinellen Lernverfahren und physikalischen Modellen, eröffnen jedoch einen neuen Weg, um die Nachteile rein datenbasierter Methoden zu umgehen und gleichzeitig ihre Vorteile zu nutzen.

Diese Arbeit schlägt hybride Methoden des maschinellen Lernens für die sichere Trajektorienplanung in kritischen Verkehrsszenarien vor. Zwei neue modellbasierte Algorithmen, nämlich Augmented CL-RRT und Augmented CL-RRT+, werden durch Erweiterung des sampling-basierten Rapid-exploring Random Tree (RRT) Algorithmus entwickelt. Diese Algorithmen planen sichere Trajektorien unter Berücksichtigung der fahrdynamischen Eigenschaften von Fahrzeugen. Sie sind probabilistisch vollständig, das heißt, dass die Wahrscheinlichkeit keine gültige Trajektorie zu finden, falls es eine gibt, asymptotisch gegen Null geht, je länger das Verfahren läuft. Daher wird ein 3D Convolutional Neural Network verwendet, um das Sampling so zu beinflussen, dass die Algorithmen schneller konvergieren. Weil die sampling-basierten Algorithmen nicht wiederholbar sind, d. h. es kann sein, dass für die gleiche Verkehrssituation unterschiedliche Trajektorien berechnet werden, wird in der Arbeit auch ein anderer hybrider Lernalgorithmus eingeführt, der generative maschinelle Lernverfahren nutzt. Es handelt sich auch um einen hybriden Lernalgorithmus, der einen mit Fahrzeugtrajektorien trainierten Variational Autoencoder mit einem deterministischen Optimierungsverfahren zum Finden sicherer Trajektorien kombiniert. Die Simulationsergebnisse mit vielen tausend kritischen Verkehrsszenarien in einer Matlab-Simulationsumgebung zeigen, dass die erforderliche Rechenzeit für hybride maschinelle Lernalgorithmen im Vergleich zu den entsprechenden modellbasierten Verfahren bei gleicher Vorhersageleistung um ein Mehrfaches reduziert wird. Diese Arbeit schlägt auch verschiedene Methoden vor, um die erforderlichen Rechenressourcen für die Umsetzung der Algorithmen in eingebetteten Systemen zu reduzieren und präsentiert Ergebnisse der Implementierung auf verschiedenen Hardware-Plattformen.

Abstract

Trajectory planning is a central task for autonomous mobility applications. A special challenge is the safe trajectory planning for vehicles in critical traffic scenarios with multiple static and dynamic objects. In a critical traffic scenario, a safe trajectory should be planned in such a way that it avoids a collision and, if this is no longer possible, it mitigates the consequences. Traditionally, model-based algorithms are used for such tasks, because an accurate understanding of the models that control actuators in safety-critical applications is important. For complex planning tasks, however, model-based algorithms are very computationally intensive and cannot be used in the vehicles, because on-board computing resources are limited.

Most machine learning algorithms, although they are usually very efficient in terms of computing resources compared to model-based approaches, are not interpretable because they are purely data-based methods. However, hybrid algorithms, a combination of machine learning methods and physical models, open up a new way to avoid the disadvantages of purely data-based methods and at the same time take advantage of their benefits.

This work proposes hybrid machine learning methods for safe trajectory planning in critical traffic scenarios. Two new model-based algorithms, namely Augmented CL-RRT and Augmented CL-RRT+, are developed by extending the sampling-based Rapid-exploring Random Tree (RRT) algorithm. These algorithms plan safe trajectories taking into account the vehicle dynamics characteristics. They are probabilistically complete, i. e., the probability of not finding a valid trajectory, if one exists, asymptotically approaches zero as long as the procedure is running. Therefore, a 3D Convolutional Neural Network is used to influence the sampling such that the algorithms converge faster. Since the sampling-based algorithms are not repeatable, i. e., it is possible that different trajectories are computed for the same traffic situation, another hybrid learning algorithm using generative machine learning is introduced in this work. It is also a hybrid learning algorithm, which combines a Variational Autoencoder trained with vehicle trajectories with a deterministic optimisation method to find safe trajectories. The simulation results with many thousands of critical traffic scenarios in a Matlab simulation environment show that the computing time required for hybrid machine learning algorithms is reduced several times compared to the corresponding model-based methods for the same prediction performance. This thesis also proposes different methods to reduce the computational resources required for the realisation of the proposed algorithms in embedded systems and presents results of their implementation on different hardware platforms.

Chapter 1

Introduction

The usage of hybrid machine learning algorithms for safe trajectory planning in critical traffic-scenarios is the main topic of this thesis. Section 1.1 describes the motivation of this work. Section 1.2 defines the aim of this work and presents the proposed methodology briefly. The set of requirements for the safe trajectory planning in particular for the vehicles in critical traffic-scenario is described in Section 1.3. Finally, Section 1.4 illustrates the outline and major contributions of this thesis.

1.1 Motivation

There is a tremendous increase in road traffic in recent years, which has created new challenges for vehicle manufacturers for transporting people and goods from one place to another quickly and most importantly, safely. It increases the risk of road fatalities. The Global status report on road safety 2018, launched by World Health Organisation in December 2018, highlights that the number of annual road traffic deaths has reached 1.35 million and it is now the leading killer of people aged 5-29 years. To ensure the reduction in road traffic fatalities, a goal of *Vision Zero* has been introduced in many countries to achieve a system with no deaths or serious injuries in road traffic.

The introduction of exteroceptive sensors like radar, camera, lidar, etc. in vehicles, has opened new possibilities to increase the comfort and safety by developing new vehicle safety functions. Adaptive Cruise Control (ACC), Lane Assistant, Autonomous Emergency Braking (AEB) are some of the vehicle comfort and safety functions that use these sensors. The introduction of such safety functions in modern vehicles has steadily reduced injury and death rates [KHB09]. Nevertheless, these functions represent only initial steps for the long term plan of *Vision Zero* goal. To achieve the this goal, a significant challenge that needs to be addressed is the collision avoidance in critical traffic-scenarios with multiple static and dynamic objects. Here, the critical traffic-scenarios mean the traffic-scenarios in which a severe collision cannot be avoided by the already available vehicle safety systems such as

AEB and the criticality measure exceeds the defined threshold. If the collision is unavoidable, then the trajectory should be modified such that the predicted severity of injury is the lowest. Such trajectory planning needs accurate dynamics modelling of road traffic-participants, a prediction algorithm for collision detection, appropriate estimations of the severity of injury and efficiency in terms of implementation so that the algorithms run in real-time. These algorithms can be part of the autonomous intervention in manually driven cars or of fully-autonomous vehicles for avoiding a collision in critical traffic-scenarios.

The importance of these algorithms can be stated from the self-driving test program of GM Cruise Chevy Bolt crash in late 2017, that was a part of the self-driving test program of General Motors, with a motorcyclist after it aborted a lane change manoeuvre [Wes]. As the Bolt initiated the lane-change manoeuvre, the gap it was trying to enter closed rapidly due to a braking lead vehicle in the adjacent lane. The motorcyclist, who was lane-splitting, moved forward beside the Bolt and blocked the return manoeuvre. The Bolt was stuck in a dilemma whether to collide with the motorcycle or to crash into a car in the adjacent lane. Here, many possibilities are available such as the merging might have been possible with a more aggressive driving style or a slightly delayed abort might have been enough time to avoid colliding with the motorcyclist. This tight interaction of decision making is still a big challenge in self-driving cars.

In order to reduce the complexity of this task, the decision-making problem for self-driving vehicles is traditionally split into two parts. The first part is the behaviour planning which plans the type of manoeuvre such as change lane, follow lane, etc. for the self-driving vehicle in the given traffic-scenario. The discretization of the behaviour is based on the assumption that the vehicle is in the semi-structured environment and performs predefined manoeuvres. It is then followed by planning the actual trajectory for the selected behaviour with low-level controllers. The selection of the manoeuvre from the predefined list of manoeuvres limits the possibilities for vehicles. This is especially not desired in critical traffic-scenarios where standard driving manoeuvres are not always sufficient to avoid the collision. On the other hand, the trajectory planning algorithm should consider maximum possible manoeuvres in the continuous state-space to increase the chances of avoiding a collision. Therefore, there is a need for developing specific algorithms for finding safe trajectories in critical traffic-scenarios.

1.2 Aim and Proposed Methodology

This work aims to research and develop efficient safe trajectory planning algorithms for vehicles in critical traffic-scenarios with multiple static and dynamic objects. Such algorithms are computationally very intensive because of the underlying complexity of the task, as well as the requirement of solving differential equations of vehicle dynamic models. Therefore, there is no algorithm for this task that works in real-time with limited onboard computational resources proposed in the literature. Also, the vehicle may enter a critical traffic-scenario where it cannot avoid a collision. In such cases, it should follow a trajectory

that will lead to a mitigation of the collision. Therefore, the trajectory planning algorithm should include an estimation of the severity of injury.

Machine learning methods offer a possibility to describe the complicated relationship between multiple input-output systems with lower computing resources. However, they are often seen as "Black-Box" as they are purely data-based methods. Some machine learning algorithms like decision trees, linear regression are interpretable, however their capacity to describe the complex relationship is limited. With the increase in the capacity of the machine learning algorithm, its interpretability decreases. As a result, they are not used in complicated safety-critical applications. However, hybrid machine learning methods, a combination of machine learning methods and physical models, open a new way to work around the disadvantages of the pure data-based methods while simultaneously exploiting their advantages.

The main objective of this work is the development of new hybrid machine learning methods for vehicle safety application and their practical implementation. The challenging task of trajectory planning for vehicles in critical traffic-scenarios is taken as the primary application for developing these methods. However, these methods can be adapted and used for trajectory planning for industrial and medical robots, drones, etc. The primary evaluation criteria for these proposed methods include *safety* and *low resource consumption*.

The basic idea of the planned combination of the machine learning algorithms and physical models in this work is symbolically shown in the Fig. 1.1. The algorithms use the predictions of other traffic-participants as the input. The sampling-based algorithms, that are most popular among many other trajectoryy planning algorithm [BIS09], for safe trajectory planning include the vehicle dynamic model as a constraint. They perform uniform or random sampling when used without machine learning algorithms. The task of machine learning algorithms is to merely assist these algorithms by learning a prior in order to bias the sampling towards promising regions that will lead to a reduction in the convergence time. This way, the final trajectory is still planned with algorithms based on vehicle

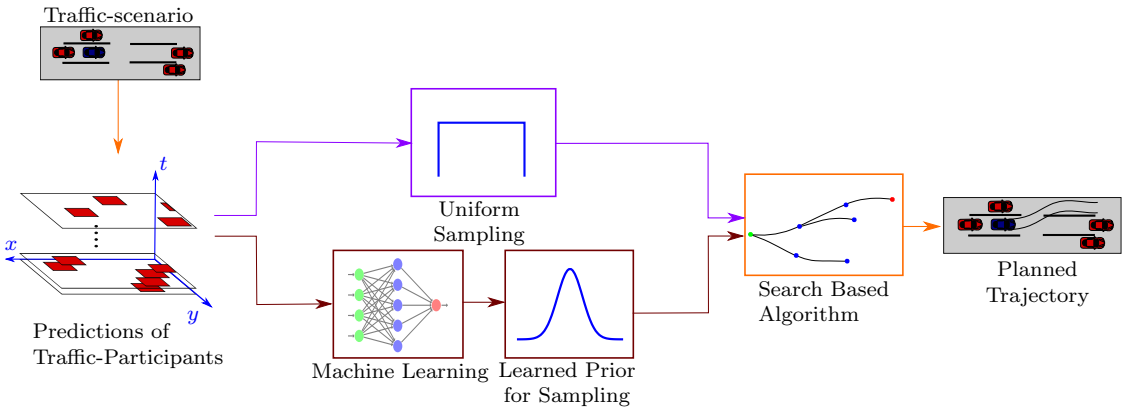


Figure 1.1: Trajectory Planning for Collision Avoidance.

dynamic models. The proposed algorithms are an effective combination of exploration property of random sampling-based search algorithms with the exploitation of gained experience learned through machine learning algorithms.

The main research results of this work include:

- Methodology for the accurate trajectory planning with vehicle dynamic models;
- Consideration of the severity of injury while planning safe trajectories;
- Methodology for hybrid machine learning methods in safety-critical applications like trajectory planning with low resource consumption;
- Evaluation of the methodology through implementation on an automotive microcontroller.

1.3 Requirement Definition for Safe Trajectory Planning

Apart from being a very complex problem, safe trajectory planning is a safety-critical application that has its particular requirements. This section lists these requirements. These requirements are divided into functional and embedded (non-functional) implementation requirements. Although the functional requirements for trajectory planning will change from one domain to another, the non-functional requirements are generally applicable to all domains.

Functional requirements describe what a system should do with regard to the desired functionality. In this work, the functional requirements for trajectory planning for vehicle safety applications are defined as follows:

- **Criticality estimation of the traffic-scenario**: In a first step, a traffic-scenario must be categorized as *critical* for the EGO vehicle in order to initiate the trajectory planning algorithm. The EGO vehicle is the vehicle in which the trajectory planning algorithm is running.
- **Computation of collision-free trajectories**: In the second step, collision-free trajectories for the EGO vehicle must be computed. Collision-free trajectories are the path followed by a vehicle as a function of time in the time interval $[t_0, t_0 + \tau]$ ($\text{e. g. } \tau = 4s$) without a collision. The EGO vehicle should be able to follow such a trajectory with a simultaneous intervention in both the lateral and longitudinal dynamics. It is because the trajectories resulting alone from full braking, as the ones that can be realized by an AEB system, are not enough to avoid collisions in many critical traffic-scenarios.
- **Prediction of the severity of injury**: For unavoidable collisions or when trajectory planning algorithms do not converge, a trajectory with low severity of collision must be computed for the collision mitigation.

- **Selection of best trajectory**: The best trajectory should be selected from all the computed trajectories based on the criteria that reflect the aspects safety (e. g. collision-free), the severity of injury (e. g. low collision speed), comfort (low accelerations during the manoeuvre), etc. The overall safety algorithm should also include a fail-safe strategy when it cannot find a collision-free or a trajectory with a nonsevere collision.

These functional requirements have been implemented in the sampling-based trajectory planning algorithms described in Chapter 3.

Embedded implementation requirements describe the properties the trajectory planning algorithms should have in order to implement them in vehicles for safety applications. They are summarized below:

- **Efficiency**: The trajectory planning algorithms should be efficient so that they should run in real-time on an automotive microcontroller.

- **Interpretability**: This means that the precise understanding of the models controlling vehicle actuators while planning and following the trajectory in critical traffic-scenarios is necessary. This requirement of interpretability arises, especially when safety-critical applications use black-box methods such as machine learning algorithms.

- **No dynamic memory allocation**: For embedded applications, especially safety-critical applications like trajectory planning in complex traffic-scenarios, dynamic memory allocation is not allowed [Hol06]. It is because many coding errors stem from mishandling of memory allocation, like attempting to allocate more memory than physically available, forgetting to free memory or continuing to use memory after it was free, etc.

- **Low variance/worst-case execution time**: The resources available on an automotive microcontroller are limited. Therefore, it is essential to know how much computational resources should be assigned for correct functional behaviour. In real-time systems, the typically used criterion is worst-case execution time. It is the maximum length of time possible for the execution of an algorithm. The worst-case execution criterion is not suitable for the trajectory planning algorithms proposed in this work are probabilistically complete, i. e., they guarantee a solution only in an infinite time. Assigning large computational resources are not recommended because they will be wasted in scenarios where less computational resources are enough. Therefore, the desirable property for a random sampling algorithm apart from high convergence speed is to have low variance in the required execution time. Apart from this, a maximum limit in terms of computational resources (time or number of samples) still have to defined.

1.4 Outline and Major Contributions of the Thesis

This thesis can be divided into three parts. The first part with Chapter 2 and Chapter 3 focusses on the theory of vehicle dynamics and sampling-based trajectory planning algorithms that include vehicle dynamic constraints, respectively. The second part, which consists of Chapters 4 and 5, deals with the theory of machine learning and its combination with sampling-based trajectory planning algorithms, respectively. The final part, i. e., Chapter 6 provides a comparative analysis of all the proposed trajectory planning algorithms and proposes further optimization techniques for reducing the required computational resources. It also summarizes the embedded implementation results of a proposed trajectory planning algorithm in various hardware platforms. The following journal papers and peer-reviewed conference papers are published as part of this thesis.

- In [CBKM16], the trajectory planning algorithm titled the Augmented CL-RRT algorithm is proposed. This algorithm uses predefined longitudinal acceleration profiles and random sampling in the lateral dynamic intervention of the vehicle to find collision-free trajectories.

- In [CBU16], the algorithm Hybrid Augmented CL-RRT, which combines a 3D convolutional neural network with the Augmented CL-RRT for faster convergence, is proposed. It also describes the method for estimating the severity of the injury within the Augmented CL-RRT algorithm.

- In [CBU17], the Augmented CL-RRT algorithm is extended with random sampling for longitudinal acceleration profiles instead of using predefined longitudinal acceleration profiles. This newly formed algorithm is named the Augmented CL-RRT+ algorithm, which is again combined with a 3D convolutional neural network for faster convergence.

- In [CBU18], an unsupervised algorithm is proposed for the generation of reference trajectories, which is then combined with a deterministic optimization algorithm for finding safe trajectories. This deterministic approach is further combined with the Augmented CL-RRT+ algorithm for planning safe trajectories.

- In [CAHBU19], machine learning based methods for replacing computationally intensive modules of trajectory planning algorithms and multiple analytical methods for reducing the computational complexity of 3D convolutional neural network are presented to make the proposed algorithms in earlier papers suitable for the implementation in an automotive microcontroller.

- In [CAHBU20], multiple alternative architectures of deep learning algorithms are presented and compared with the 3D convolutional neural network.

1.5 Notations

Throughout this thesis, vectors and matrices are denoted by lower and upper case bold letters, respectively. Random variables are written using sans serif fonts. The symbol "*" denotes element-wise multiplication, $\odot$ represents Hadamard product, $\otimes$ is the symbol for the outer product, $E_{\mathbf{x}}$ is the expectation with respect to $\mathbf{x}$, $||\ .\ ||$ is the norm of a vector and $|\ .\ |$ represents the absolute value. The symbol ∇ indicates the gradient while $\boldsymbol{e}_{\{.\}}$ is the unit vector along the corresponding axis in the subscript. $\mathcal{O}$ is the Landau symbol. A list of the most important symbols that are used in the thesis can be found in Appendix 8.2.5. Expressions are emphasized by writing them in italic type.

Chapter 2

Dynamic Models and Controllers

Dynamic models are widely used in control and trajectory planning algorithms to approximate the behaviour of road traffic-participants in response to control actions. A high fidelity model may accurately reflect the response of the vehicle, but the added detail complicates the trajectory planning. Therefore, many trajectory planning algorithms use only models with sufficient details to find an optimal solution. These models work well in normal operating conditions. However, they are not accurate enough with harsh control actions that are usually required in critical traffic-scenarios. An accurate behaviour is necessary in critical traffic-scenarios as an inaccurate behaviour due to the low fidelity model might lead to a fatal collision. Therefore, the compromise concerning the fidelity of the vehicle dynamic model in critical traffic-scenarios, which is the focus of this work, is not recommended.

The outline of this chapter is as follows: Section 2.1 presents the most commonly used models of road traffic-participants followed by their validation procedure and results described in Section 2.2. Section 2.3 proposes longitudinal and lateral dynamic controllers for the road traffic-participants. Finally, Section 2.4 illustrates the self-developed simulation environment in which all these dynamic models and controllers are implemented. The purpose of this simulation environment is to simulate traffic-scenarios and generate data necessary for training the machine learning algorithms described in Chapter 5.

2.1 Dynamic Models

This section describes different dynamic models for the road traffic-participants.

2.1.1 Vehicle Coordinate Frame

Vehicle modelling begins with the notion of vehicle configuration, that represents its pose or position. The equations of motion for vehicle dynamics are usually expressed in a local

vehicle orthogonal Cartesian coordinate frame $B(C\text{XYZ})$, attached to the vehicle at the center of gravity C as shown in the Fig. 2.1. The X-axis is the longitudinal axis passing through C and directed towards forward direction of the vehicle. The Y-axis goes laterally to the left from the viewpoint of the driver and the Z-axis is perpendicular to both X- and Y-axis opposite to the gravitational acceleration g on a flat road. Vehicle orientation is shown with three angles: roll angle about the X-axis, pitch angle about the Y-axis, and yaw angle about the Z-axis as shown in the Fig. 2.2. However, the vehicle is considered as a rigid body and the dynamics of the rigid body is considered as a planar motion [Jaz08]. The planar model is applicable whenever the forward, lateral and yaw velocities are important and are enough to examine the behaviour of a vehicle. Another essential angle related to vehicle dynamics is the sideslip angle β. It is an angle the velocity $\boldsymbol{v}$ of the vehicle makes with the X-axis of the vehicle about the Z-axis.

The position and orientation of the vehicle coordinate frame $B(C\text{XYZ})$ is measured with respect to the grounded fixed orthogonal Cartesian coordinate frame $G(O\text{XYZ})$. The vehicle coordinate frame is called the *body or local coordinate frame* and the grounded frame is called the *global or inertial coordinate frame*. The analysis of vehicle motion is equivalent to expressing the position and orientation of the body coordinate frame in a

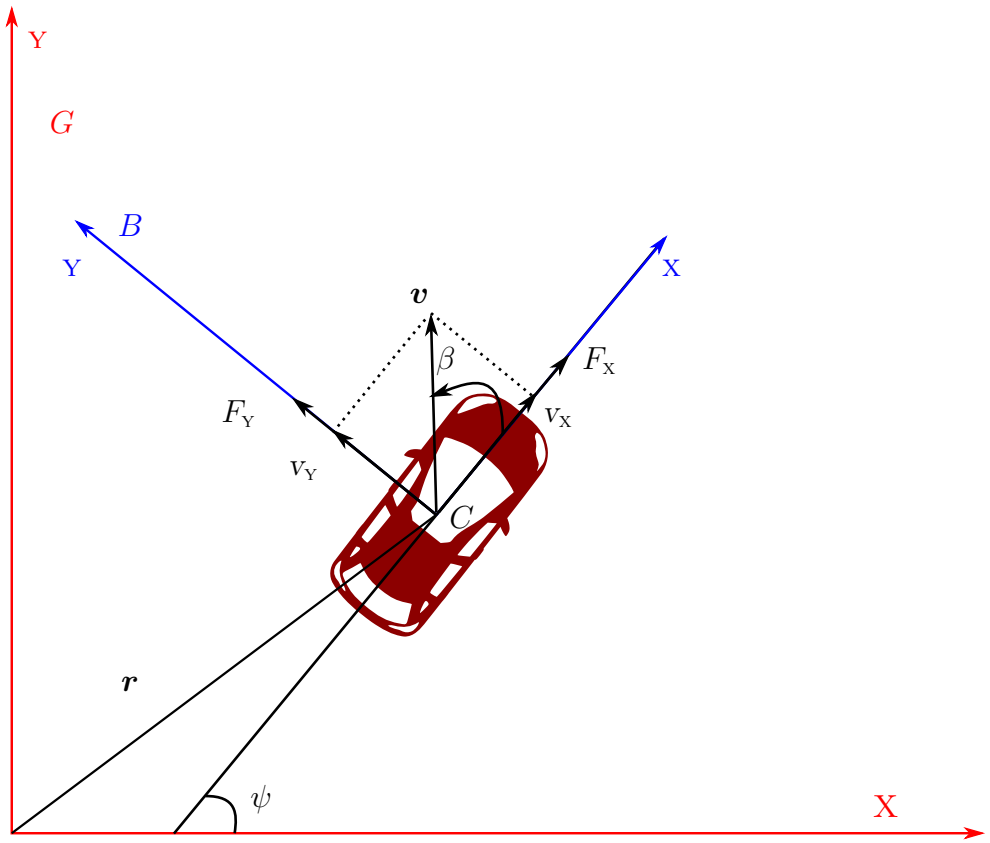


Figure 2.1: Vehicle Coordinate Frames.

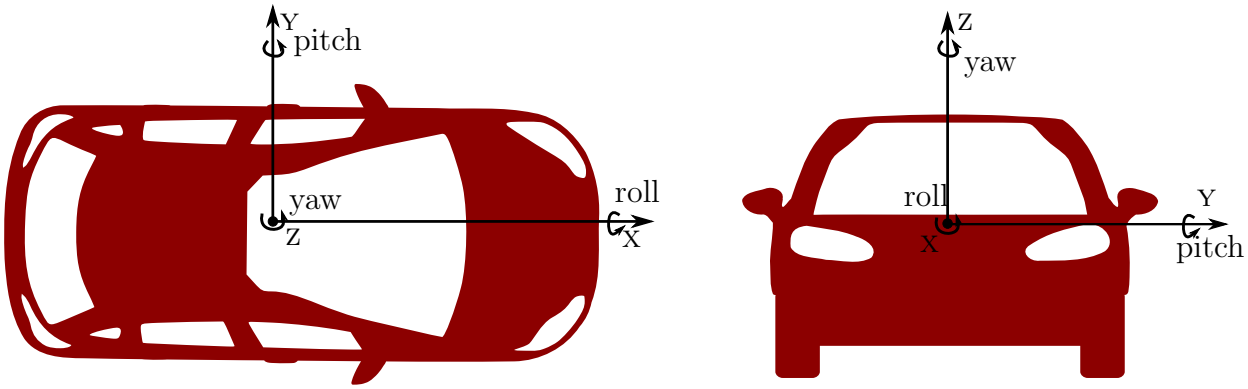


Figure 2.2: Six Degrees of Freedom for the Vehicle.

global coordinate frame [Jaz08].

2.1.2 Rigid Body Dynamics

The principles of rigid body dynamics, as well as the Newton-Euler equations of motion that describe the translational and rotational motion of the rigid body under the action of external forces, are reviewed in this section. The Newton-Euler equations of motion are further used to derive differential equations of vehicle dynamics.

Derivative Transformation Formula

The analysis of the body coordinate frame movement in the global coordinate frame describes the vehicle dynamics. Therefore, it is necessary to find a relation between two relatively moving coordinate frames. This relation expressing the transformation of derivative from one coordinate frame to the other is called as the *derivative transformation formula*. The time derivative of a vector depends on the coordinate frame in which the derivative is taken. For example, the time derivative of a position vector $\boldsymbol{r}_P$ of the point P on a rigid body in the global coordinate frame is denoted as

$$\frac{{}^{G}d}{dt}\boldsymbol{r}_P. \tag{2.1}$$

The left superscript on the derivative symbol indicates the frame in which the derivative is taken. The position vector $\boldsymbol{r}_P$ can be expressed in the body and global coordinate frames as

$${}^{B}\boldsymbol{r}_P = \mathrm{X}\boldsymbol{e}_\mathrm{X} + \mathrm{Y}\boldsymbol{e}_\mathrm{Y} + \mathrm{Z}\boldsymbol{e}_\mathrm{Z}, \tag{2.2}$$

and

$${}^{G}\boldsymbol{r}_P = \mathrm{X}\boldsymbol{e}_\mathrm{X} + \mathrm{Y}\boldsymbol{e}_\mathrm{Y} + \mathrm{Z}\boldsymbol{e}_\mathrm{Z}, \tag{2.3}$$

respectively. Here, $(\boldsymbol{e}_\mathrm{X}, \boldsymbol{e}_\mathrm{Y}, \boldsymbol{e}_\mathrm{Z})$ and $(\boldsymbol{e}_\mathrm{X}, \boldsymbol{e}_\mathrm{Y}, \boldsymbol{e}_\mathrm{Z})$ are the unit vectors along the coordinate axes of the body and the global coordinate frame, respectively. The time derivative of

a vector is simple if the vector is expressed in the same coordinate frame in which the derivative is taken, because the unit vectors are constant and scalar coefficients are the only time variables. Therefore, the derivative of ${}^{B}\boldsymbol{r}_P$ in body coordinate frame B is

$$\frac{{}^{B}d}{dt}{}^{B}\boldsymbol{r}_P = {}^{B}\boldsymbol{v}_P = \dot{\mathrm{X}}\boldsymbol{e}_{\mathrm{X}} + \dot{\mathrm{Y}}\boldsymbol{e}_{\mathrm{Y}} + \dot{\mathrm{Z}}\boldsymbol{e}_{\mathrm{Z}} \tag{2.4}$$

and the derivative of ${}^{G}\boldsymbol{r}_P$ in global coordinate frame G is

$$\frac{{}^{G}d}{dt}{}^{G}\boldsymbol{r}_P = {}^{G}\boldsymbol{v}_P = \dot{\mathrm{X}}\boldsymbol{e}_{\mathrm{X}} + \dot{\mathrm{Y}}\boldsymbol{e}_{\mathrm{Y}} + \dot{\mathrm{Z}}\boldsymbol{e}_{\mathrm{Z}}. \tag{2.5}$$

However, when the point P is moving in frame B while B is rotating in G, the derivative of the ${}^{B}\boldsymbol{r}_P$ in the global coordinate frame G is

$$\frac{{}^{G}d}{dt}{}^{B}\boldsymbol{r}_P = \frac{{}^{G}d}{dt}(\mathrm{X}\boldsymbol{e}_{\mathrm{X}} + \mathrm{Y}\boldsymbol{e}_{\mathrm{Y}} + \mathrm{Z}\boldsymbol{e}_{\mathrm{Z}}), \tag{2.6}$$

$$= \dot{\mathrm{X}}\boldsymbol{e}_{\mathrm{X}} + \dot{\mathrm{Y}}\boldsymbol{e}_{\mathrm{Y}} + \dot{\mathrm{Z}}\boldsymbol{e}_{\mathrm{Z}} + \mathrm{X}\frac{{}^{G}d}{dt}\boldsymbol{e}_{\mathrm{X}} + \mathrm{Y}\frac{{}^{G}d}{dt}\boldsymbol{e}_{\mathrm{Y}} + \mathrm{Z}\frac{{}^{G}d}{dt}\boldsymbol{e}_{\mathrm{Z}}. \tag{2.7}$$

Using the expression from Eq. 2.4, the above equation is changed to

$$\frac{{}^{G}d}{dt}{}^{B}\boldsymbol{r}_P = \frac{{}^{B}d}{dt}{}^{B}\boldsymbol{r}_P + \mathrm{X}\frac{{}^{G}d}{dt}\boldsymbol{e}_{\mathrm{X}} + \mathrm{Y}\frac{{}^{G}d}{dt}\boldsymbol{e}_{\mathrm{Y}} + \mathrm{Z}\frac{{}^{G}d}{dt}\boldsymbol{e}_{\mathrm{Z}}. \tag{2.8}$$

As the velocity of any rotating vector with fixed length is the cross product of instantaneous angular velocity ${}^{B}_{G}\boldsymbol{\omega}_B$ and the vector itself, the above equation can be converted as

$$\frac{{}^{G}d}{dt}{}^{B}\boldsymbol{r}_P = \frac{{}^{B}d}{dt}{}^{B}\boldsymbol{r}_P + \mathrm{X}{}^{B}_{G}\boldsymbol{\omega}_B \times \boldsymbol{e}_{\mathrm{X}} + \mathrm{Y}{}^{B}_{G}\boldsymbol{\omega}_B \times \boldsymbol{e}_{\mathrm{Y}} + \mathrm{Z}{}^{B}_{G}\boldsymbol{\omega}_B \times \boldsymbol{e}_{\mathrm{Z}}, \tag{2.9}$$

$$= \frac{{}^{B}d}{dt}{}^{B}\boldsymbol{r}_P + {}^{B}_{G}\boldsymbol{\omega}_B \times (\mathrm{X}\boldsymbol{e}_{\mathrm{X}} + \mathrm{Y}\boldsymbol{e}_{\mathrm{Y}} + \mathrm{Z}\boldsymbol{e}_{\mathrm{Z}}), \tag{2.10}$$

$$= \frac{{}^{B}d}{dt}{}^{B}\boldsymbol{r}_P + {}^{B}_{G}\boldsymbol{\omega}_B \times {}^{B}\boldsymbol{r}_P. \tag{2.11}$$

This result is utilized to define the generalized transformation of the differential operator from body to global coordinate frame such that

$$\frac{{}^{G}d}{dt}{}^{B}\square = {}^{B}_{G}\square_P = \frac{{}^{B}d}{dt}{}^{B}\square + {}^{B}_{G}\boldsymbol{\omega}_B \times {}^{B}\square\,. \tag{2.12}$$

The final result ${}^{B}_{G}\square_P$ shows the global time derivative expressed in the body frame. The vector $\square$ might be any vector. This equation is called the derivative transformation formula and it relates the time derivative of a vector as it would be seen from global frame G to its derivative as seen in frame B. This equation can be applied to any vector for derivative transformation between two relatively moving coordinate frames.

Newton-Euler Dynamics for Rigid Vehicle

The Newton-Euler equations of motion describe the combined translational and rotational dynamics of a rigid body. It is a relation between the motion of the center of gravity of the rigid body and the sum of forces and torques acting on a rigid body. The application of a force system is emphasized by Newton's second law of motion, which is also called *Newton's equation of motion*. It states that the global rate of change of linear momentum ${}^{G}\boldsymbol{p}$ is equal to the global applied force ${}^{G}\boldsymbol{F}$ and mathematically it is defined as

$$ {}^{G}\boldsymbol{F} = \frac{{}^{G}d}{dt}\,{}^{G}\boldsymbol{p} = \frac{{}^{G}d}{dt}\,(m\,{}^{G}\boldsymbol{v}_B), \tag{2.13} $$

where ${}^{G}\boldsymbol{v}_B$ is the velocity of the rigid body expressed in the global coordinate frame. Applying the derivative transformation formula from Eq. 2.12, the force can be expressed in the body coordinate frame B as

$$ {}^{B}_{G}\boldsymbol{F} = m\,{}^{B}_{G}\dot{\boldsymbol{v}}_B + m\,{}^{B}_{G}\boldsymbol{\omega}_B \times {}^{B}_{G}\boldsymbol{v}_B. \tag{2.14} $$

Using this equation, the X, Y and Z components of the force ${}^{B}_{G}\boldsymbol{F}$ can be calculated as

$$ \begin{bmatrix} F_{\mathrm{X}} \\ F_{\mathrm{Y}} \\ F_{\mathrm{Z}} \end{bmatrix} = \begin{bmatrix} m\dot{v}_{\mathrm{X}} + m(\omega_{\mathrm{Y}} v_{\mathrm{Z}} - \omega_{\mathrm{Z}} v_{\mathrm{Y}}) \\ m\dot{v}_{\mathrm{Y}} + m(\omega_{\mathrm{X}} v_{\mathrm{Z}} - \omega_{\mathrm{Z}} v_{\mathrm{X}}) \\ m\dot{v}_{\mathrm{Z}} + m(\omega_{\mathrm{X}} v_{\mathrm{Y}} - \omega_{\mathrm{Y}} v_{\mathrm{X}}) \end{bmatrix}. \tag{2.15} $$

Similarly, the components of the applied moment on the vehicle can be calculated by extending the second law of motion to include a rotational motion. This extension is called *Euler's equation of motion*. It states that the global rate of change of angular momentum ${}^{G}\boldsymbol{L}$ is equal to global applied moment ${}^{G}\boldsymbol{M}$. Mathematically, it is represented as

$$ {}^{G}\boldsymbol{M} = \frac{{}^{G}d}{dt}\,{}^{G}\boldsymbol{L}. \tag{2.16} $$

Again applying the derivative transformation formula from Eq. 2.12, the applied moment in body coordinate frame can be expressed as

$$ {}^{B}_{G}\boldsymbol{M} = {}^{B}\dot{\boldsymbol{L}} + {}^{B}_{G}\boldsymbol{\omega}_B \times {}^{B}\boldsymbol{L}. \tag{2.17} $$

The angular momentum of a rigid body rotating about an axis passing through the origin of the local reference frame is in fact the product of the inertia tensor of the object and the angular velocity. Therefore, by replacing angular momentum ${}^{B}\boldsymbol{L}$ with ${}^{B}\boldsymbol{I}\,{}^{B}_{G}\boldsymbol{\omega}_B$, where ${}^{B}\boldsymbol{I}$ isth inertia tensor, the above expression becomes

$$ {}^{B}_{G}\boldsymbol{M} = {}^{B}\boldsymbol{I}\,{}^{B}_{G}\dot{\boldsymbol{\omega}}_B + {}^{B}_{G}\boldsymbol{\omega}_B \times {}^{B}\boldsymbol{I}\,{}^{B}_{G}\boldsymbol{\omega}_B. \tag{2.18} $$

Also, it can be assumed that the body-coordinate frame is the principal coordinate frame of vehicle so that it has a diagonal matrix for the moment of inertia such that

$$ {}^{B}\boldsymbol{I} = \begin{bmatrix} I_{\mathrm{X}} & 0 & 0 \\ 0 & I_{\mathrm{Y}} & 0 \\ 0 & 0 & I_{\mathrm{Z}} \end{bmatrix}, \tag{2.19} $$

where I_{X}, I_{Y} and I_{Z} are the moment of inertia of the vehicle around X-, Y- and Z-axis, respectively. Therefore, the X, Y and Z components of the applied moment ${}^{B}_{G}\boldsymbol{M}$ are

$$\begin{bmatrix} M_{\mathrm{X}} \\ M_{\mathrm{Y}} \\ M_{\mathrm{Z}} \end{bmatrix} = \begin{bmatrix} I_{\mathrm{X}}\dot{\omega}_{\mathrm{X}} - (I_{\mathrm{Z}} - I_{\mathrm{Y}})\omega_{\mathrm{Y}}\omega_{\mathrm{Z}} \\ I_{\mathrm{Y}}\dot{\omega}_{\mathrm{Y}} - (I_{\mathrm{X}} - I_{\mathrm{Z}})\omega_{\mathrm{X}}\omega_{\mathrm{Z}} \\ I_{\mathrm{Z}}\dot{\omega}_{\mathrm{Z}} - (I_{\mathrm{X}} - I_{\mathrm{Y}})\omega_{\mathrm{X}}\omega_{\mathrm{Z}} \end{bmatrix}. \tag{2.20}$$

As the vehicle is considered a rigid body with planar motion, only three degrees of freedom are considered. This includes the translational velocity along the X- and Y-axis and the rotational velocity around the Z-axis. Therefore, apart from the lateral and longitudinal translational forces F_{X} and F_{Y} and the torque M_{Z}, all other components of the applied force and moment are considered zero. With these assumptions the force, momentum, velocity, angular velocity and angular acceleration vectors become

$${}^{B}\boldsymbol{F} = \begin{bmatrix} F_{\mathrm{X}} \\ F_{\mathrm{Y}} \\ 0 \end{bmatrix}, {}^{B}\boldsymbol{M} = \begin{bmatrix} 0 \\ 0 \\ M_{\mathrm{Z}} \end{bmatrix}, {}^{B}\dot{\boldsymbol{v}} = \begin{bmatrix} \dot{\mathrm{v}}_{\mathrm{X}} \\ \dot{\mathrm{v}}_{\mathrm{Y}} \\ 0 \end{bmatrix}, {}^{B}_{G}\boldsymbol{\omega}_{B} = \begin{bmatrix} 0 \\ 0 \\ \omega_{\mathrm{Z}} \end{bmatrix}, {}^{B}_{G}\dot{\boldsymbol{\omega}}_{B} = \begin{bmatrix} 0 \\ 0 \\ \dot{\omega}_{\mathrm{Z}} \end{bmatrix}. \tag{2.21}$$

Substituting the vectors from Eq. 2.21 and Eq. 2.19 in Eq. 2.15 and Eq. 2.20, the equations of motion for the vehicle are converted into

$$F_{\mathrm{X}} = m\dot{\mathrm{v}}_{\mathrm{X}} - m\dot{\psi}\mathrm{v}_{\mathrm{Y}}, \tag{2.22}$$

$$F_{\mathrm{Y}} = m\dot{\mathrm{v}}_{\mathrm{Y}} + m\dot{\psi}\mathrm{v}_{\mathrm{X}}, \tag{2.23}$$

$$M_{\mathrm{Z}} = \ddot{\psi}I_{\mathrm{Z}}. \tag{2.24}$$

2.1.3 Nonlinear Two-Track Model

In this section, a nonlinear two-track vehicle dynamic model in a planar motion is described. A two-track vehicle dynamic model assumes that the vehicle is a rigid body and incorporates the effects of all individual tire forces and moments on the vehicle. Fig. 2.3 presents the planar view of a vehicle that shows important parameters of the vehicle necessary to describe the nonlinear two-track model. The vehicle coordinate system has three unit vectors $\boldsymbol{e}_{\mathrm{X}}$, $\boldsymbol{e}_{\mathrm{Y}}$ and $\boldsymbol{e}_{\mathrm{Z}}$. The vehicle center of gravity is denoted by C and is modelled to lie on the road plane. The vehicle has a track-width w and the wheel-base ℓ. The steering angle of the wheels with respect to the longitudinal axis of the vehicle are δ_i and the forces $F_{\mathrm{X}i}$ and $F_{\mathrm{Y}i}$ are acting on each wheel, where the subscript $i \in \{\mathrm{fl}, \mathrm{fr}, \mathrm{rl}, \mathrm{rr}\}$ indicates whether it is the front left (fl), front right (fr), rear left(rl), or rear right(rr).

Derivation of Differential Equations

The two-track vehicle dynamic model is described by three coupled differential equations, where the state variables are the velocity v, the sideslip angle β and yaw rate $\dot{\psi}$. These

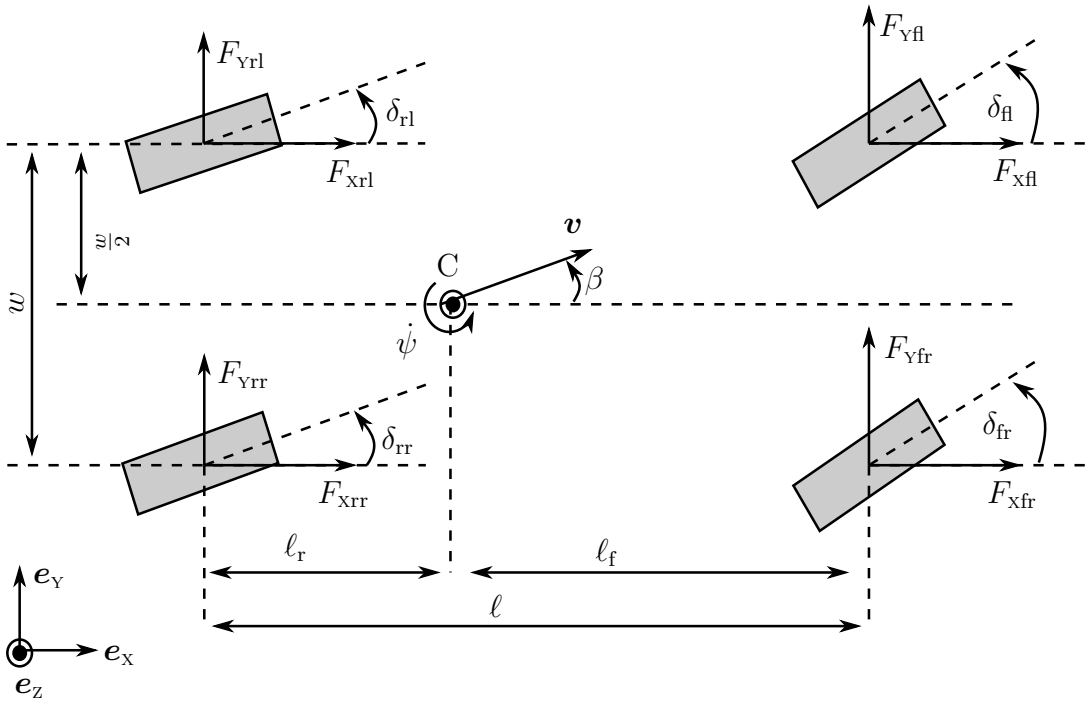


Figure 2.3: Planar View of the Vehicle Adapted from [Bot17].

differential equations can be formulated from the forces and moments acting on the rigid vehicle. Using the expressions from Eq. 2.22, 2.23 and 2.24, these forces can be expressed in the body coordinate frame B as

$$\sum_i F_{\mathrm{X}i} = m\dot{v}_{\mathrm{X}} - m\dot{\psi}v_{\mathrm{Y}}, \tag{2.25}$$

$$\sum_i F_{\mathrm{Y}i} = m\dot{v}_{\mathrm{Y}} + m\dot{\psi}v_{\mathrm{X}}, \tag{2.26}$$

$$\sum_i {}^{B}_{G}\boldsymbol{r}_i \times {}^{B}_{G}\boldsymbol{F}_i = I_{\mathrm{z}G}{}^{B}_{G}\dot{\boldsymbol{\omega}}_B \tag{2.27}$$

where, $\sum_i F_{\mathrm{X}i}$ and $\sum_i F_{\mathrm{Y}i}$ are the sum of X- and Y-components of all forces acting on the vehicle, respectively. The position vector ${}^{B}_{G}\boldsymbol{r}_i$ is the lever arm for the force ${}^{B}_{G}\boldsymbol{F}_i$ with respect to center of gravity C of the vehicle.

By substituting the values for v_{X} and v_{Y} as $v\cos(\beta)$ and $v\sin(\beta)$, the Eq. 2.25 and 2.26 become

$$\sum_i F_{\mathrm{X}i} = m\left(\dot{v}\cos(\beta) - v(\dot{\beta}+\dot{\psi})\sin(\beta)\right), \tag{2.28}$$

$$\sum_i F_{\mathrm{Y}i} = m\left(\dot{v}\sin(\beta) + v(\dot{\beta}+\dot{\psi})\cos(\beta)\right), \tag{2.29}$$

respectively. Resolving $\dot{v}$ and $\dot{\beta}$ by multiplying Eq. 2.28 with $\cos(\beta)$ and Eq. 2.29 with $\sin(\beta)$ followed by their addition, the resulting equation is

$$\cos(\beta)\sum_i F_{\mathrm{X}i} + \sin(\beta)\sum_i F_{\mathrm{Y}i} = m\dot{v}\left(\sin^2(\beta) + \cos^2(\beta)\right), \tag{2.30}$$

which upon simplification becomes

$$\dot{v} = \frac{1}{m}\left(\cos(\beta)\sum_i F_{\mathrm{X}i} + \sin(\beta)\sum_i F_{\mathrm{Y}i}\right). \tag{2.31}$$

Substituting $\dot{v}$ in Eq. 2.29, the derivative of the sideslip angle becomes

$$\dot{\beta} = \frac{1}{mv}\left(cos(\beta)\sum_i F_{\mathrm{X}i} - sin(\beta)\sum_i F_{\mathrm{Y}i}\right) - \dot{\psi}. \tag{2.32}$$

The equation for the yaw rate can also be derived from Eq. 2.27. The left-hand side of the Eq. 2.27 is equivalent to adding the moments resulting from the applied forces on each tire. The moment can be calculated using the product of applied force and its position vector. By individually calculating the moment acting on each tire and subsequently adding, the total moment acting on the vehicle can be calculated. Substituting this in Eq. 2.25, the equation for the yaw rate results into

$$I_{\mathrm{z}}\ddot{\psi} = \Big(\ell_{\mathrm{f}}(F_{\mathrm{Yfl}} + F_{\mathrm{Yfr}}) + \frac{w}{2}(F_{\mathrm{Xfr}} - F_{\mathrm{Xfl}}) - \ell_{\mathrm{r}}(F_{\mathrm{Yrl}} + F_{\mathrm{Yrr}}) + \frac{w}{2}(F_{\mathrm{Xrr}} - F_{\mathrm{Xrl}})\Big), \tag{2.33}$$

$$\ddot{\psi} = \frac{1}{I_{\mathrm{z}}}\Big(\ell_{\mathrm{f}}(F_{\mathrm{Yfl}} + F_{\mathrm{Yfr}}) + \frac{w}{2}(F_{\mathrm{Xfr}} - F_{\mathrm{Xfl}}) - \ell_{\mathrm{r}}(F_{\mathrm{Yrl}} + F_{\mathrm{Yrr}}) + \frac{w}{2}(F_{\mathrm{Xrr}} - F_{\mathrm{Xrl}})\Big). \tag{2.34}$$

The X- and Y-components of all forces acting on the vehicle are the addition of respective components of forces acting on each tire, if other forces stemming from wind, spoilers, etc. are ignored. Therefore, they are described as

$$\sum_i F_{\mathrm{X}i} = F_{\mathrm{Xfl}} + F_{\mathrm{Xfr}} + F_{\mathrm{Xrl}} + F_{\mathrm{Xrr}}, \tag{2.35}$$

$$\sum_i F_{\mathrm{Y}i} = F_{\mathrm{Yfl}} + F_{\mathrm{Yfr}} + F_{\mathrm{Yrl}} + F_{\mathrm{Yrr}}. \tag{2.36}$$

The forces $F_{\mathrm{X}ij}$ and $F_{\mathrm{Y}ij}$ acting on individual tire can be expressed as

$$F_{\mathrm{Xij}} = F_{\ell\mathrm{ij}}\cos(\delta_{ij}) - F_{\mathrm{sij}}\sin(\delta_{ij}), \tag{2.37}$$

$$F_{\mathrm{Yij}} = F_{\ell\mathrm{ij}}\sin(\delta_{ij}) + F_{\mathrm{sij}}\cos(\delta_{ij}), \tag{2.38}$$

where F_{sij} is the side force acting on the tire parallel to the rotational axis of the tire while $F_{\ell\mathrm{ij}}$ is the longitudinal force acting on the tire in the forward direction perpendicular to

the F_{sij}. Using the Eq. 2.35, 2.36, 2.37, 2.38, the first differential equation, Eq. (2.31), can be rewritten as

$$\begin{aligned}\dot{v} =& \frac{1}{m}\Big(F_{\ell\text{fl}}\cos(\delta_{fl})\cos(\beta) - F_{\text{sfl}}\sin(\delta_{fl})\cos(\beta) + F_{\ell\text{fr}}\cos(\delta_{fr})\cos(\beta) - F_{\text{sfr}}\sin(\delta_{fr})\cos(\beta) \\ &+ F_{\ell\text{rl}}\cos(\delta_{rl})\cos(\beta) - F_{\text{srl}}\sin(\delta_{rl})\cos(\beta) + F_{\ell\text{rr}}\cos(\delta_{rr})\cos(\beta) - F_{\text{srr}}\sin(\delta_{rr})\cos(\beta) \\ &+ F_{\ell\text{fl}}\sin(\delta_{fl})\sin(\beta) + F_{\text{sfl}}\cos(\delta_{fl})\sin(\beta) + F_{\ell\text{fr}}\sin(\delta_{fr})\sin(\beta) + F_{\text{sfr}}\cos(\delta_{fr})\sin(\beta) \\ &+ F_{\ell\text{rl}}\sin(\delta_{rl})\sin(\beta) + F_{\text{srl}}\cos(\delta_{rl})\sin(\beta) + F_{\ell\text{rr}}\sin(\delta_{rr})\sin(\beta) + F_{\text{srr}}\cos(\delta_{rr})\sin(\beta)\Big). \end{aligned} \tag{2.39}$$

Eq. 2.39 can be simplified by taking into account that
$\cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta) = \cos(\alpha-\beta) - \sin(\alpha)\sin(\beta) + \sin(\alpha)\sin(\beta) = \cos(\alpha-\beta)$ and
$-\sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta) = -\sin(\alpha-\beta) - \cos(\alpha)\sin(\beta) + \cos(\alpha)\sin(\beta)$ such that

$$\begin{aligned}\dot{v} =& \frac{1}{m}\Big(F_{\ell\text{fl}}\cos(\delta_{fl}-\beta) + F_{\ell\text{fr}}\cos(\delta_{fr}-\beta) - F_{\text{sfl}}\sin(\delta_{fl}-\beta) \\ &- F_{\text{sfr}}\sin(\delta_{fr}-\beta) + F_{\ell\text{rl}}\cos(\delta_{rl}-\beta) + F_{\ell\text{rr}}\cos(\delta_{rr}-\beta) \\ &- F_{\text{srl}}\sin(\delta_{rl}-\beta) - F_{\text{srr}}\sin(\delta_{rr}-\beta)\Big) \end{aligned} \tag{2.40}$$

By performing similar steps, the second differential equation, i. e., Eq. 2.32, can also be reformulated as

$$\begin{aligned}\dot{\beta} =& \frac{1}{mv}\Big(F_{\ell\text{fl}}\sin(\delta_{fl}-\beta) + F_{\ell\text{fr}}\sin(\delta_{fr}-\beta) + F_{\text{sfl}}\cos(\delta_{fl}-\beta) \\ &+ F_{\text{sfr}}\cos(\delta_{fr}-\beta) + F_{\ell\text{rl}}\sin(\delta_{rl}-\beta) + F_{\ell\text{rr}}\sin(\delta_{rr}-\beta) \\ &+ F_{\text{srl}}\cos(\delta_{rl}-\beta) + F_{\text{srr}}\cos(\delta_{rr}-\beta)\Big) - \dot{\psi}. \end{aligned} \tag{2.41}$$

Finally, the third differential equation, Eq. 2.34, can be rewritten, using the Eq. 2.37 and Eq. 2.38, as

$$\begin{aligned}\ddot{\psi} =& \frac{1}{I_z}\Big(\ell_f\Big(F_{\ell\text{fl}}\sin(\delta_{fl}) + F_{\ell\text{fr}}\sin(\delta_{fr}) + F_{\text{sfl}}\cos(\delta_{fl}) + F_{\text{sfr}}\cos(\delta_{fr})\Big) \\ &+ \frac{w}{2}\Big(F_{\ell\text{fr}}\cos(\delta_{fr}) - F_{\ell\text{fl}}\cos(\delta_{fl}) - F_{\text{sfr}}\sin(\delta_{fr}) + F_{\text{sfl}}\sin(\delta_{fl})\Big) \\ &- \ell_r\Big(F_{\ell\text{rl}}\sin(\delta_{rl}) + F_{\ell\text{rr}}\sin(\delta_{rr}) + F_{\text{srl}}\cos(\delta_{rl}) + F_{\text{srr}}\cos(\delta_{rr})\Big) \\ &+ \frac{w}{2}\Big(F_{\ell\text{rr}}\cos(\delta_{rr}) - F_{\ell\text{rl}}\cos(\delta_{rl}) - F_{\text{srr}}\sin(\delta_{rr}) + F_{\text{srl}}\sin(\delta_{rl})\Big)\Big) \end{aligned} \tag{2.42}$$

The three equations 2.40, 2.41 and 2.42 describe the quantities v, β and $\dot{\psi}$ which are used to estimate the vehicle dynamics. Therefore, the state vector for the vehicle dynamics can be expressed as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{v} \\ \dot{\beta} \\ \ddot{\psi} \end{bmatrix} = \boldsymbol{f}(v, \beta, \dot{\psi}, F_{\ell\text{ij}}, F_{\text{sij}}, \delta_{\text{ij}}). \tag{2.43}$$

Tire Dynamics

From the state vector in Eq. 2.43, that describes the vehicle dynamics, it is clear that knowledge about the forces arising from the interaction between the tires and the road surface is of vital importance to evaluate the dynamics of the vehicle. The force transmission between tire and road surface takes place through the friction between tire contact patch and road. Therefore, the amount of the force which can be transferred is dependent on the frictional properties of the tire and roadway. This section explains the procedure for the calculation of these forces.

Brush Tire Model

The *Brush Tire Model* [Pac06] explains the creation of forces between the road and tire. This model considers that a part of the contact area of the tire to the ground show sliding friction while another part of the contact area shows adhesion at the same time. The formation of the longitudinal forces can be described through the shear deformation of tire profile elements and the friction behaviour between the tire and road. The profile element, while transporting through the contact patch, adheres to the road surface for a certain time. This creates a speed difference, which results in a longitudinal slip. As long as the tire adheres to the road surface, the profile element gets distorted. Assuming that the distortion acts like a linear spring, the locally transmitted longitudinal force can be calculated and the integration of the force on the whole contact patch will give the total longitudinal force. The relative velocity between the tire and surface, that is the speed the distortion of profile elements sets in, is responsible for the longitudinal forces. Indirectly, the longitudinal slip can be used to determine the longitudinal forces on the tire $F_{\ell \mathrm{ij}}$. Fundamentally one can assume that the profile elements exhibit similar deformation characteristics in the lateral direction as well that results in lateral forces on tire F_{sij}.

The slip for one direction can be defined as the velocity difference between transversal and rotational velocity for that direction divided by a reference velocity. Longitudinal slip describes the state of motion of a driven, braked or non-driven rolling vehicle. In an ideal tire, the velocity of the contact patch should be equal to the tire speed. In a real tire, this is not the case. For a driven (accelerated) wheel, the transversal velocity $v_{\ell ij}$ is smaller than rotational velocity $\omega_{ij} r_{ij}$ of the tire. In this case, the longitudinal slip is

$$s_{\ell ij} = \frac{\omega_{ij} r_{ij} - v_{\ell ij}}{\omega_{ij} r_{ij}}. \tag{2.44}$$

On the other hand, the longitudinal slip for a decelerating wheel, for which the translational velocity $v_{\ell ij}$ is higher than rotational velocity $\omega_{ij} r_{ij}$ of the tire, is

$$s_{\ell ij} = \frac{v_{\ell ij} - \omega_{ij} r_{ij}}{v_{\ell ij}}. \tag{2.45}$$

Summarizing both Eq. 2.44 and Eq. 2.45 one can write

$$s_{\ell ij} = \frac{\omega_{ij} r_{ij} - v_{\ell ij}}{\max(|v_{\ell ij}|, |\omega_{ij} r_{ij})|}, \tag{2.46}$$

where ω_{ij} is the angular velocity and r_{ij} the radius of the of the ij^{th} tire. A positive slip indicates wheels are spinning while negative slip indicates they are skidding.

The lateral slip of a tire is the sideways motion of a tire which occurs when the sideways forces of tire are greater than its frictions resistance. This can occur, for instance, in cornering. The side slip is defined as [Pac06]

$$s_{sij} = \frac{v_{sij}}{v_{\ell ij}}. \tag{2.47}$$

The expression for the lateral slip is derived from the Fig. 2.4. A tire that is acted upon by the lateral force F_{sij} gets a velocity component v_{sij} lateral to the rolling direction. The angle formed between the longitudinal axis and the velocity of the tire $\boldsymbol{v}_{ij}$ at the contact point is called as slip angle α_{ij}. Hence, according to the Fig. 2.4

$$tan(\alpha_{ij}) = \frac{v_{sij}}{v_{lij}}, \tag{2.48}$$

where, v_{sij} and v_{lij} are the lateral and longitudinal component of the velocity of the tire to the rolling direction, respectively. The variable $tan(\alpha_{ij})$ is also know as the lateral slip or skew slip. In normal driving condition $|\alpha_{ij}| < 12^0$.

Calculation of Slip Values

Longitudinal slip cannot be measured. Therefore, it is estimated by Eq. 2.46. The slip angles α_{ij} for all tires, required for calculating the lateral slip, can be calculated geometrically

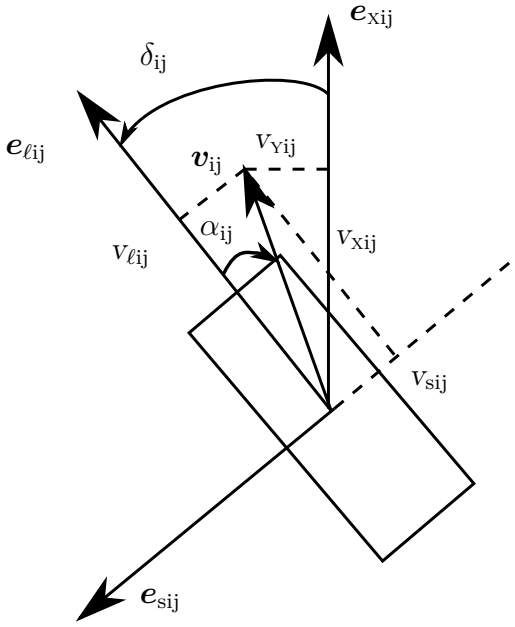


Figure 2.4: Velocities for the ij^{th} Tire Adapted from [Bot17].

from Fig. 2.4 as follows:

$$\tan(\delta_{\mathrm{ij}} - \alpha_{\mathrm{ij}}) = \frac{v_{\mathrm{Yij}}}{v_{\mathrm{Xij}}}, \tag{2.49}$$

$$\alpha_{\mathrm{ij}} = \delta_{\mathrm{ij}} - \arctan\left(\frac{v_{\mathrm{Yij}}}{v_{\mathrm{Xij}}}\right). \tag{2.50}$$

By applying the derivative transformation formula, the velocity $\boldsymbol{v}_{ij}$ can be calculated as

$$\boldsymbol{v}_{ij} = \boldsymbol{v} + \boldsymbol{\omega} \times \boldsymbol{r}_{\mathrm{C,ij}}, \tag{2.51}$$

where $\boldsymbol{r}_{\mathrm{C,ij}}$ is the radius of the rotation for the ij^{th} tire with respect to the center of gravity of the vehicle and $\boldsymbol{v}$ is the velocity vector in the center of gravity. From this expression, the components of velocity $v_{\mathrm{X}ij}$ and $v_{\mathrm{Y}ij}$ are calculated for each tire and substituted in Eq. 2.50. The resulting expressions for slip angles of all tires are

$$\alpha_{\mathrm{fl}} = \delta_{\mathrm{fl}} - \arctan\left(\frac{v\sin(\beta) + \ell_{\mathrm{f}}\dot{\psi}}{v\cos(\beta) - \frac{w}{2}\dot{\psi}}\right), \tag{2.52}$$

$$\alpha_{\mathrm{fr}} = \delta_{\mathrm{fr}} - \arctan\left(\frac{v\sin(\beta) + \ell_{\mathrm{f}}\dot{\psi}}{v\cos(\beta) + \frac{w}{2}\dot{\psi}}\right), \tag{2.53}$$

$$\alpha_{\mathrm{rl}} = \delta_{\mathrm{rl}} - \arctan\left(\frac{v\sin(\beta) - \ell_{\mathrm{r}}\dot{\psi}}{v\cos(\beta) - \frac{w}{2}\dot{\psi}}\right), \tag{2.54}$$

$$\alpha_{\mathrm{rr}} = \delta_{\mathrm{rr}} - \arctan\left(\frac{v\sin(\beta) - \ell_{\mathrm{r}}\dot{\psi}}{v\cos(\beta) + \frac{w}{2}\dot{\psi}}\right). \tag{2.55}$$

Magic Tire Formula

For small slip values, both the longitudinal force F_{lij} and the side force F_{sij} can be modelled to grow linearly with the corresponding slip. If the slip gets larger the static friction changes into a sliding friction for an increasing number of profile elements located in the area of the tire that touches the road and the resulting force between the road and the tire exceeds the maximum possible force F_{max} which corresponds to the maximum adhesion coefficient μ_{max}. At very high slip values, the transferred force decreases to the value F_{slide}, which occurs during sliding and remains constant. Fig. 2.5 shows this nonlinear mapping between the slip and forces.

The *Magic Tire Formula* [Pac06] is designed to provide a realistic tire characteristics curve that can easily fit to measured data by adjusting some parameters. This is an empirical formula as it does not have any particular physical basis. The simplified Magic Tire formulae that give good approximated mapping of longitudinal and lateral forces of tire to their corresponding slips are

$$F_{\ell ij} = F_{\mathrm{z}ij}\mu_{\ell ij}\sin\left(c_{\ell ij}\arctan\left(b_{\ell ij}\frac{s_{\ell ij}}{\mu_{\ell ij}}\right)\right), \tag{2.56}$$

$$F_{sij} = F_{\mathrm{z}ij}\mu_{sij}\sin\left(c_{sij}\arctan\left(b_{sij}\frac{s_{sij}}{\mu_{sij}}\right)\right), \tag{2.57}$$

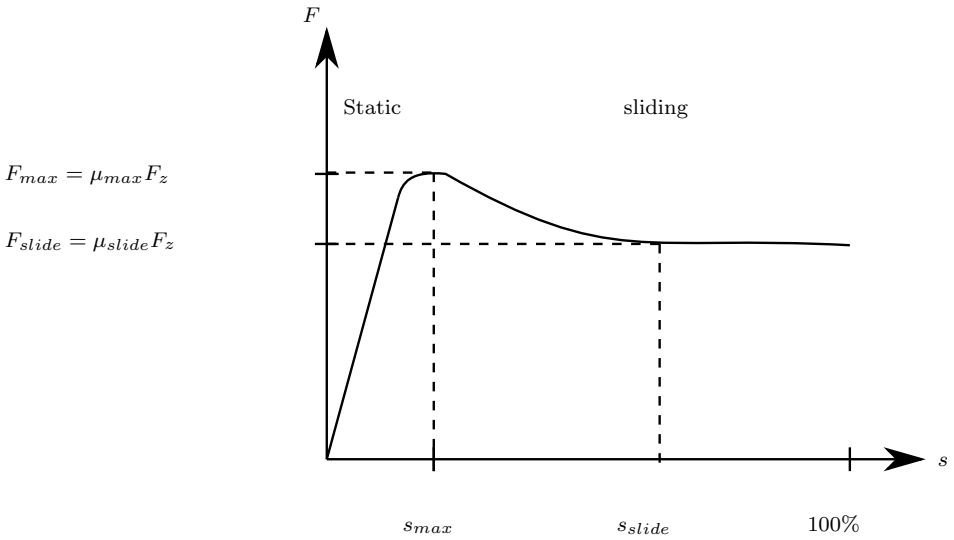


Figure 2.5: Force as a Function of Slip Adapted from [Bot17].

where F_{zij} is the vertical force on the ij^{th} tire while $\mu_{\ell ij}$ and μ_{sij} are the friction coefficients in the longitudinal and lateral direction of the tire, respectively. The tire parameters $c_{\ell ij}, b_{\ell ij}, c_{sij}, b_{sij}$ are constants that can be adapted to fit a specific tire.

Superposition of Longitudinal and Lateral Forces

Pure longitudinal forces will be transmitted by tire only during purely straight line driving and pure lateral forces will be transmitted by tire only during cornering with constant velocity. In general driving situations, the longitudinal and lateral forces are overlapped. When longitudinal and side forces appear at the same time the computation must be adapted to take into account the magnitude of the resulting force. According to the Coulomb law, this resulting force must be smaller than $\mu_{max} F_{zij}$ such that

$$\sqrt{F_{\ell ij}^2 + F_{sij}^2} < \mu_{max} F_{zij}, \tag{2.58}$$

This means the maximum transmissible lateral force decreases with the application of a longitudinal force. In order to consider the superposition effects during driving situations, in which both longitudinal and lateral slip occur, an absolute slip s_{aij} and and angle $\Psi_a ij$ are introduced as

$$s_{aij} = \sqrt{s_\ell^2 + tan(\alpha_{ij}^2)}, \tag{2.59}$$

$$\Psi_{aij} = arctan\left(\frac{tan(\alpha_i j)}{s_{lij}}\right). \tag{2.60}$$

The absolute force F_{aij} in direction Ψ_{aij} is then calculated as

$$F_{aij} = \sqrt{\frac{s_{lij}^2}{s_{aij}^2} F_{lij}^2 + \frac{(tan(\alpha_{ij}))^2}{s_{aij}^2} F_{sij}^2}. \tag{2.61}$$

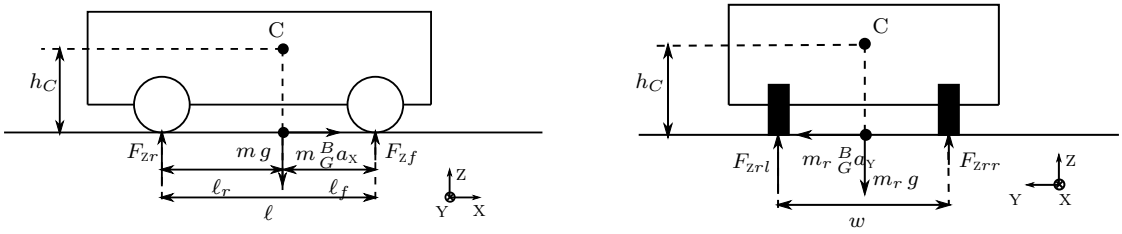
The computation of resulting longitudinal forces $F_{lij}^{(new)}$ and lateral forces $F_{sij}^{(new)}$ represented as

$$F_{\ell ij}^{(new)} = F_{aij}\cos(\Psi_{aij}), \tag{2.62}$$

$$F_{sij}^{(new)} = F_{aij}\sin(\Psi_{aij}). \tag{2.63}$$

Vertical Forces and Accelerations

The vertical force F_{zij}, required for calculating the longitudinal and lateral forces using Eq. 2.56 and 2.57, depends on the accelerations in X- and Y-direction of the vehicle and the weight of the vehicle. Acceleration and deceleration of the vehicle generate a pitch moment, whereas driving in a curve generates a roll moment. The vertical forces F_{zij} on the vehicle axle are influenced by these moments. Therefore, these forces also need to be taken into consideration while modelling vehicle dynamics. Fig. 2.6a illustrates the vertical forces acting on the tires due to a longitudinal acceleration/deceleration ${}_G^B a_X$ while Fig. 2.6b indicates the vertical forces acting on the tires due to the lateral acceleration ${}_G^B a_Y$ generated while turning on a left curve.



a) Vertical Forces due to ${}_G^B a_X$.

b) Vertical Forces due to Turning (Left Curve).

Figure 2.6: Forces on Tires Depend on X- and Y-Acceleration Adapted from [Bot17].

The moments acting on the vehicles in the Fig. 2.6a can be balanced in center of gravity C of the vehicle such that

$$F_{zr}\ell_r - F_{zf}\ell_f - h_c m\,{}_G^B a_X = 0, \tag{2.64}$$

where F_{zr} and F_{zf} are the total vertical force acting on the rear and front axle of the vehicle, respectively. Rearranging the Eq. 2.64, the expression for the force F_{zr} can be obtained as

$$F_{zr} = \frac{\ell_f}{\ell_r}F_{zf} + \frac{h_C}{\ell_r}m\,{}_G^B a_X. \tag{2.65}$$

Also, the vertical forces in Fig. 2.6a can be balanced to the weight of the vehicle to get

$$F_{zr} + F_{zf} = mg. \tag{2.66}$$

By combining the Eq. 2.65 and Eq. 2.66, the force F_{zr} can also be expressed as

$$F_{zr} = \frac{\ell_f}{\ell_r}mg - \frac{\ell_f}{\ell_r}F_{zr} + \frac{h_C}{\ell_r}m\,{}_G^B a_X, \tag{2.67}$$

$$F_{zr} = \frac{\ell_f}{\ell}mg + \frac{h_C}{\ell}m\,{}_G^B a_X. \tag{2.68}$$

Substituting the Eq. 2.68 in Eq. 2.66, the force F_{zf} can be calculated as

$$F_{zf} = mg - F_{zr}, \tag{2.69}$$

$$= \frac{\ell_r}{\ell} mg - \frac{h_C}{\ell} m \, {}^{B}_{G}a_{X}. \tag{2.70}$$

The Eq. 2.68 and Eq. 2.70 can be rewritten as

$$F_{zr} = \left(\frac{\ell_f}{\ell} m + \frac{h_C}{\ell} \frac{{}^{B}_{G}a_{X}}{g} \right) g, \tag{2.71}$$

$$F_{zf} = \left(\frac{\ell_r}{\ell} mg - \frac{h_C}{\ell} \frac{{}^{B}_{G}a_{X}}{g} \right) g, \tag{2.72}$$

respectively. From these equations the virtual masses for front and rear axle can be defined as

$$m_f = m \left(\frac{\ell_r}{\ell} - \frac{h_C}{\ell} \frac{{}^{B}_{G}a_{X}}{g} \right), \tag{2.73}$$

$$m_r = m \left(\frac{\ell_f}{\ell} + \frac{h_C}{\ell} \frac{{}^{B}_{G}a_{X}}{g} \right), \tag{2.74}$$

respectively. Similar steps can be performed on the Fig. 2.6b. The moments from Fig. 2.6b can also be balanced in center of gravity C such that

$$F_{zrl} \frac{w}{2} - F_{zrr} \frac{w}{2} + h_c m_r \, {}^{B}_{G}a_{Y} = 0, \tag{2.75}$$

$$F_{zrl} = F_{zrr} - \frac{2h_C}{w} m_r \, {}^{B}_{G}a_{Y}. \tag{2.76}$$

Also, balancing of vertical forces acting on the vehicle from Fig. 2.6b gives

$$F_{zrl} + F_{zrr} = m_r g. \tag{2.77}$$

By combining the Eq. 2.76 and Eq. 2.77, the force F_{zrl} can be expressed as

$$F_{zrl} = m_r g - F_{zrl} - m_r \frac{2h_C}{w} \, {}^{B}_{G}a_{Y} \tag{2.78}$$

$$= m_r \left(\frac{1}{2} g - \frac{h_C}{w} \, {}^{B}_{G}a_{Y} \right). \tag{2.79}$$

By substituting the expression for m_r from Eq. 2.74, the force F_{zrl} becomes

$$F_{zrl} = m \left(\frac{\ell_f}{\ell} + \frac{h_C}{\ell} \frac{{}^{B}_{G}a_{X}}{g} \right) \left(\frac{1}{2} g - \frac{h_C}{w} \, {}^{B}_{G}a_{Y} \right). \tag{2.80}$$

In a similar way, the other tire forces in z-direction can be computed as

$$F_{zrr} = m \left(\frac{\ell_f}{\ell} + \frac{h_C}{\ell} \frac{{}^{B}_{G}a_{X}}{g} \right) \left(\frac{1}{2} g + \frac{h_C}{w} \, {}^{B}_{G}a_{Y} \right), \tag{2.81}$$

$$F_{zfl} = m\left(\frac{\ell_r}{\ell} - \frac{h_C}{\ell}\frac{{}^{B}_{G}a_{\mathrm{X}}}{g}\right)\left(\frac{1}{2}g - \frac{h_C}{w}\,{}^{B}_{G}a_{\mathrm{Y}}\right), \tag{2.82}$$

$$F_{zfr} = m\left(\frac{\ell_r}{\ell} - \frac{h_C}{\ell}\frac{{}^{B}_{G}a_{\mathrm{X}}}{g}\right)\left(\frac{1}{2}g + \frac{h_C}{w}\,{}^{B}_{G}a_{\mathrm{Y}}\right). \tag{2.83}$$

The accelerations ${}^{B}_{G}a_{\mathrm{X}}$ and ${}^{B}_{G}a_{\mathrm{Y}}$ are available from sensors in vehicles, but they can also be computed using Eq. 2.28 and Eq. 2.29 such that

$${}^{B}_{G}a_x = \left(\dot{v}\cos(\beta) - \mathrm{v}\left(\dot{\beta} + \dot{\psi}\right)\sin(\beta)\right), \tag{2.84}$$

$${}^{B}_{G}a_y = \left(\dot{v}\sin(\beta) + \mathrm{v}\left(\dot{\beta} + \dot{\psi}\right)\cos(\beta)\right). \tag{2.85}$$

The vertical forces acting on each tire of the vehicle, calculated using the Eq. **??**, can be introduced in the Magic-Tire formula in Eq. EqMagicTire to compute the forces on the tires depending on the accelerations acting on the vehicle.

2.1.4 Single-Track Kinematic Model

Single-Track Kinematic Model is a vehicle dynamic model, shown in Fig. 2.7 which considers constraints such as

$$\dot{\mathrm{X}} = v\cos(\theta, \tag{2.86}$$

$$\dot{\mathrm{Y}} = v\sin(\theta), \tag{2.87}$$

$$\dot{\theta} = \frac{v}{\ell}\tan(\delta), \tag{2.88}$$

$$a_{\mathrm{X},min} \leq a_{\mathrm{X}} \leq a_{\mathrm{X},max}, \tag{2.89}$$

$$\parallel \delta \parallel \leq \delta_{max}, \tag{2.90}$$

$$\parallel \dot{\delta} \parallel \leq \dot{\delta}_{max}. \tag{2.91}$$

The input to the model are steering angle δ and the longitudinal acceleration a_{X}. The maximum steering angle and maximum slew rate is given by δ_{max} and $\dot{\delta}_{max}$, respectively. $a_{\mathrm{X},min}$ and $a_{\mathrm{X},max}$ are the minimum and maximum bounds of the longitudinal acceleration.

2.1.5 Decoupled x- and y-Dynamics

The assumption for this dynamic model is that the dynamics in X- and Y-directions are decoupled. Therefore, it is used for pedestrian movement simulation only. The dynamic equations of motion of this model are

$$\dot{v_{\mathrm{X}}}(t) = a_{\mathrm{X}}(t), \tag{2.92}$$

$$\dot{\mathrm{X}}(t) = v_{\mathrm{X}}(t), \tag{2.93}$$

$$\dot{v_{\mathrm{Y}}} = a_{\mathrm{Y}}(t), \tag{2.94}$$

$$\dot{\mathrm{Y}}(t) = v_{\mathrm{Y}}(t). \tag{2.95}$$

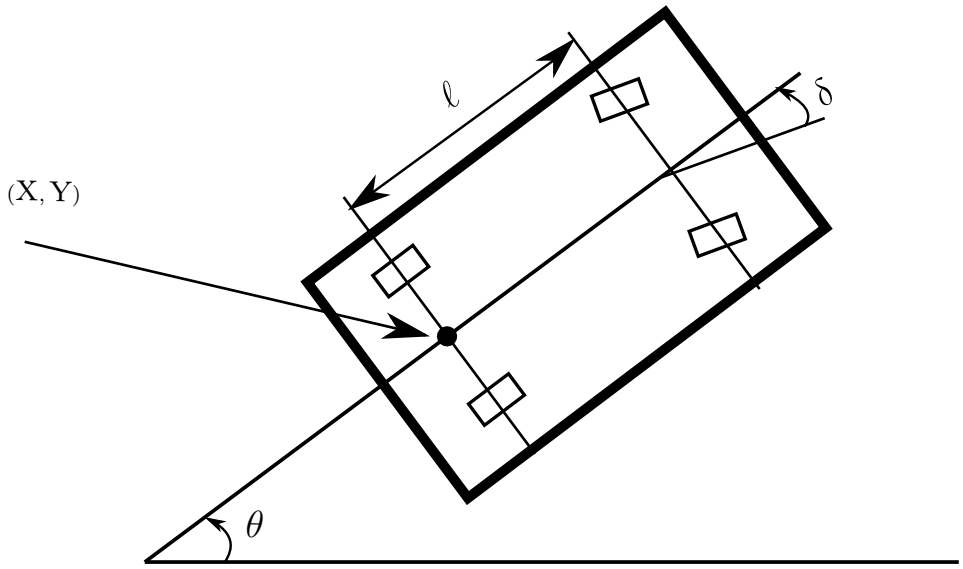


Figure 2.7: Single-Track Kinematic Model.

2.2 Validation of Vehicle Dynamic Models

In the literature, many vehicle dynamic models ranging from simple models like the single-track kinematic model to complex models like the nonlinear two-track model, are proposed. Although simple models require low computation time, they are not accurate enough, especially for the harsh vehicle manoeuvres required in critical traffic-scenarios. Therefore, it is necessary to investigate different vehicle dynamic models and their suitability for the given problem.

The vehicle dynamic models need to be validated against the real vehicle data for two reasons. Firstly, to find the parameter values which cannot be directly measured, such as parameters for Magic Tire Formula and, secondly for evaluating the accuracy of the models. Therefore, the data for different manoeuvres with a vehicle, a Audi A6 Avant (2013), was measured on the test-track. The vehicle was equipped with external GPS sensors to determine the position of the vehicle during the entire manoeuvre. In addition, access to the vehicle CAN bus made it possible to measure vehicle data, such as speed, acceleration, steering angle, from internal sensors. All technical specifications of the Audi A6 Avant such as dimensions, center of gravity locations, which is necessary for vehicle dynamic models are also measured. These details are mentioned in 8.1. The parameters used for the Magic Tire Formula, as well as other parameters required in the two-track model, such as the coefficient of friction, are estimated with the help of the measured data for one manoeuvre and subsequently validated against measurement data for other manoeuvres.

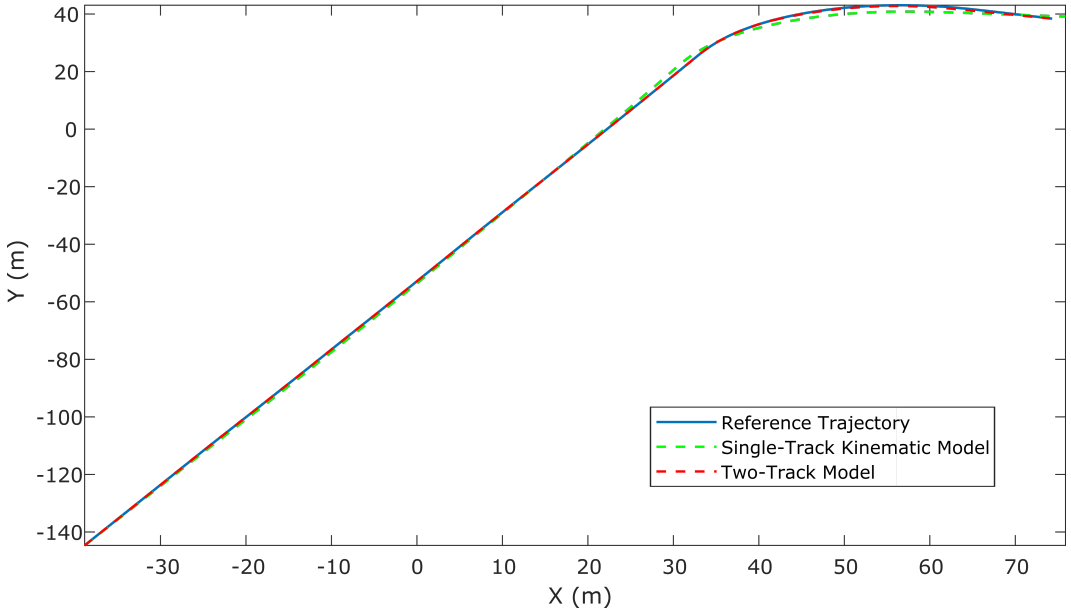


Figure 2.8: Step Steering Input Manoeuvres.

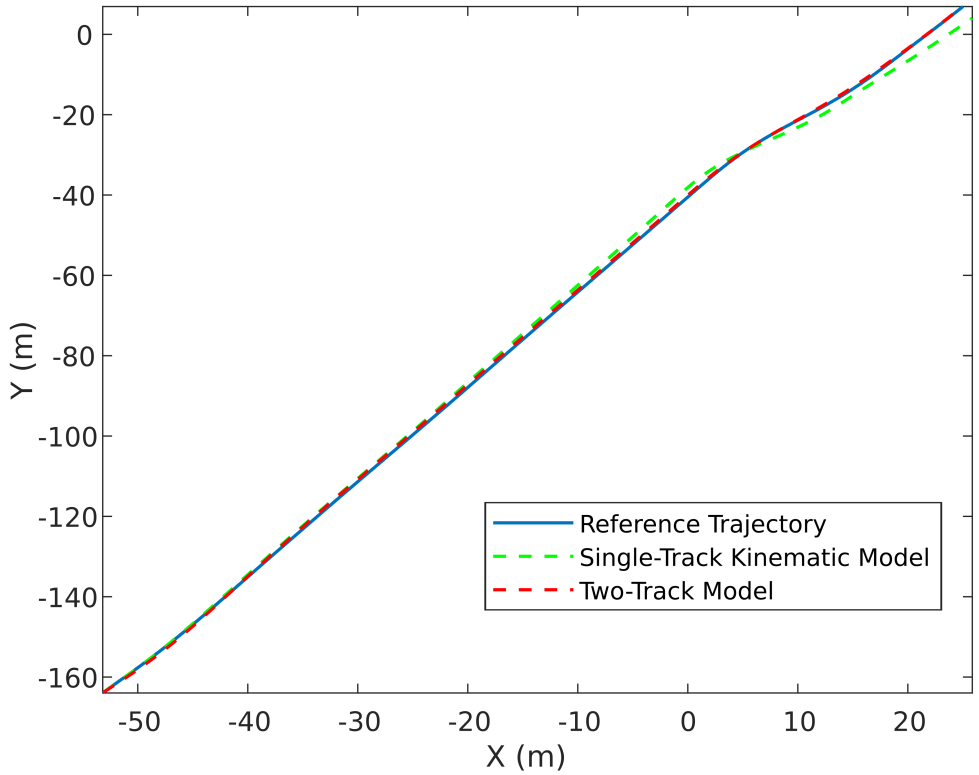


Figure 2.9: Double Lane Change Manoeuvres.

For the validation of the dynamic models, two manoeuvres are analysed: *step steering input* and *double lane change*. During the step steering input manoeuvre, the vehicle moves with constant velocity with zero steering angle input and at a specific time instant the steering angle value is changed abruptly to a constant value. On the other hand, in double lane change manoeuvre, as the name suggests, the vehicle does a double lane change with a constant velocity. The unknown parameters are estimated using data from a step steering input manoeuvre and then validated against all available data. As input data for the geometric model and the two-track model, the start position (X- and Y- coordinates) of the vehicle, as well as speed and steering angle over a period of time, are used. The vehicle position from the simulation is compared with the reference vehicle position measured by GPS sensors. An example of the simulation results for each of the manoeuvres is shown in Fig. 2.8 and 2.9. The graphs show the result of the nonlinear two-track model and the single-track model with comparison to the reference trajectory (GPS measurements). This comparison shows that, as expected, the two-track model is more accurate than the single-track model. Therefore, it is decided to use the two-track model as the basis for safe trajectory planning.

2.3 Dynamic Controllers

This section describes controllers for vehicles to follow a desired trajectory. The task of the controllers is to calculate the inputs, namely steering angle and vehicle speed, for the vehicle models. The calculation of the steering angle values for a curved path is dependent on the vehicle speed. Therefore, the longitudinal dynamic controller calculates the desired vehicle speed which is followed by the lateral dynamic controller that computs the steering angle values. These inputs for the vehicle motion, i. e., the vehicle speed and the steering angle values are calculated at every time-step successively, so that they adapt to the newly encountered situations.

These controllers are applicable for both nonlinear two-track model and single-track kinematic model. Therefore, they are used for the simulation and control of the EGO vehicle, other vehicles as well as for the bicycles.

2.3.1 Longitudinal Dynamic Controller

To drive in urban traffic, a vehicle has to adapt its speed to the other road users, ensure a safe distance to other traffic participants, and perform lane changes and turning manoeuvres. An essential part of the vehicle control system is a longitudinal controller for vehicle acceleration and deceleration. A goal of the vehicle longitudinal dynamic controller in many critical traffic-scenarios is to reduce the relative velocity to zero when the distance between the vehicle and the collision object reaches a predefined value of $d_{desired}$. This change in velocity of the vehicle is controlled by changing the longitudinal slip values of the four tires $S_{l,fl}, S_{l,fr}, S_{l,rl}, S_{l,rr}$ for the nonlinear two-track model and by directly calculating the value of longitudinal acceleration a_x in case of the single-track kinematic model.

Assuming that at time instance t, the distance between the collision object and the EGO vehicle is $d(t)$, the goal of the controller is to reach the relative velocity of zero after a certain time. In order to achieve this smoothly, the controller calculates a desired relative velocity $v_{rel,desired}$ at every instance. The value of $v_{rel,desired}$ is influenced by the current and the desired distance between the vehicle and the collision object $d(t)$ and $d_{desired}$, respectively. It is calculated with a P-controller with an expression

$$v_{rel,desired}(t) = K_v(d(t) - d_{desired}), \tag{2.96}$$

where K_v is the tuned proportionality gain. The slip values required for attaining the $v_{rel,desired}(t)$ are calculated by another P-controller such that

$$S_{l,fl} = -K_l(v_{rel}(t) - v_{rel,desired}(t)), \tag{2.97}$$

where K_l is the proportionality gain. The other slip values , $S_{l,fr}, S_{l,rl}, S_{l,rr}$ are also calculated with the same equation.

2.3.2 Lateral Dynamic Controller

In order to follow the desired trajectory, the steering angle has to be changed based on the desired trajectory curve, the lateral deviation between the vehicle position and the desired trajectory, and also the velocity of the vehicle. Human drivers preview the path and change the steering angle accordingly. The change in the steering angle is different at different speeds. In order to follow the desired trajectory, human drivers start steering from an earlier position when the velocity of the vehicle is higher and vice versa. This is an intuition behind the *pure pursuit controller* [Cou92]. The pure pursuit control is a method of geometrically determining the curvature that will drive the vehicle to a chosen reference point $\boldsymbol{p}_{ref}$. It is a point on the path that is one look-ahead distance from the current vehicle position. An arc that joins the current point and the goal point is constructed. The chord length of this arc is the look-ahead distance.

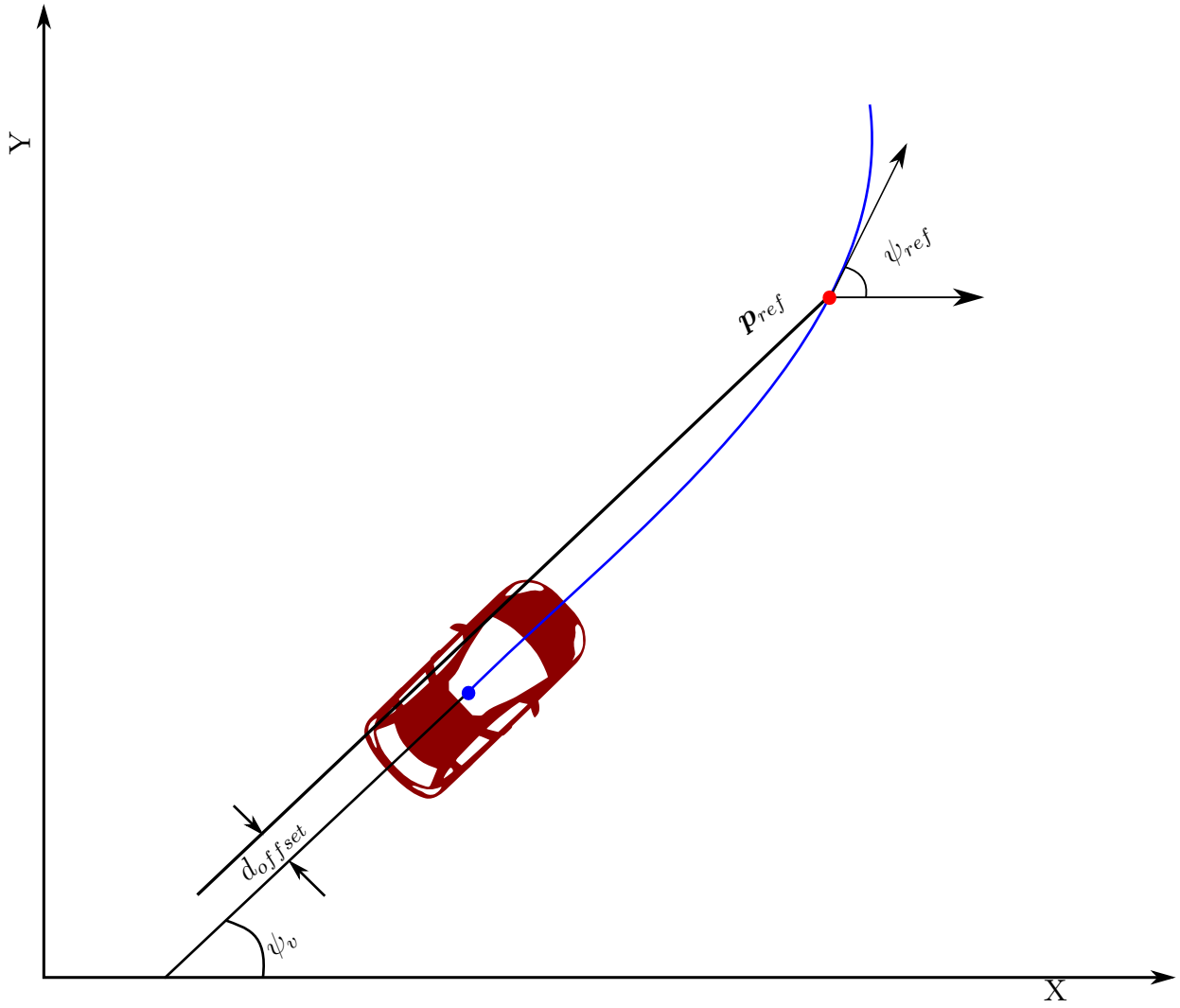


Figure 2.10: Lateral Dynamic Controller.

A predictive controller is developed to follow a trajectory given a current speed $\boldsymbol{v}(t)$ using the concept of the pure pursuit controller. The difference is in the definition of the look-ahead distance. As the input is the reference trajectory to this controller, the look-ahead distance is calculated along the reference trajectory to find the goal point $\boldsymbol{p}_{ref}$ and no arc needs to be constructed. Also, it is adaptive based on the current speed $\boldsymbol{v}(t)$ of the vehicle. Fig. 2.10 illustrates the functionality of this controller. Initially, the controller finds the path point (x', y') closest to the center of gravity of the vehicle and then defines the reference point $\boldsymbol{p}_{ref}$ along the reference trajectory at a look-ahead distance from this point. Further it calculates the yaw angle ψ_{ref} at the reference point. Finally, two P-controllers for following deviations are used to calculate the steering angle values:

- The offset d_{offset} between the actual location of the center of gravity of the vehicle and the reference position $\boldsymbol{p}_{ref}$,

- The difference between the yaw angle of the vehicle ψ at the current position and the yaw angle of the trajectory at reference point ψ_{ref}.

2.4 Simulation Environment

Humans can learn many remarkably complex skills that exceed the proficiency and robustness of even the most sophisticated robots. However, they also draw on a lifetime of experience, learning multiple years how to interact with the world around us. Requiring such a lifetime of experience for a learning-based autonomous system is quite demanding as the they would need to operate continuously, autonomously, and initially at a low level of proficiency before they become useful. This is especially impossible in safety-critical situations like autonomous driving. The creation of separate infrastructure for testing of autonomous vehicles is also costly. Therefore, safety-critical autonomous systems are trained initially in simulation environments until they get a minimum level of proficiency before putting them in a test or real environment.

Simulating many years of autonomous systems interaction is quite feasible with modern parallel computing, physics simulation, and rendering technology. Moreover, the resulting data comes with automatically-generated annotations. The challenge with simulated training is that even the best available simulators do not perfectly capture reality. Models trained purely on synthetic data fail to generalize to the real world, as there is a discrepancy between simulated and real environments, in terms of both visual and physical properties. In fact, the more the fidelity of simulations, the more efforts have to be spent in order to build them.

The difficulty of transferring simulated experience into the real world is often called the "reality gap". This reality gap prevents to repeat the simulated robotic performance into effective real-world performance. Visual perception often constitutes the most significant part of this reality gap. In this work, the proposed algorithms make abstraction from perception layer, i. e, it considers sensor processing is already done and it receives complete information of the surrounding. Here, the reality gap that needs to be considered is between simulated vehicle dynamics and real vehicle dynamics. Therefore, it is necessary to validate the vehicle dynamic models used for the trajectory planning algorithm. This can be achieved by validating the vehicle dynamic model used in simulation with real vehicle data, as explained in Section 2.2.

For this work, a simulation environment *TrafficSim* [Ami18] has been developed from scratch in MATLAB. The reason for developing and using an own simulation environment is that it provides full control over the use of vehicle models and controllers and it is the basis for devoloping algorithms that are designed to run on automotive microcontrollers. Since one of the final goals is the embedded implementation of the algorithms, the vehicle models and controllers can also be ported to an automotive microcontroller. Also, the simulation environment is developed such that the traffic-scenario generation is easy and

can be automated. This is very useful for machine learning algorithms, which need a huge amount of data. As the trajectory planning algorithms described in Chapter 3 are also implemented in the simulation environment, the data is also automatically annotated and labelled. The simulation environment uses the validated models for the simulation, which is essential for training the machine learning algorithms.

Summary

In the first part of this chapter, the dynamic models for different road traffic-participants are described. Later, these models were validated against real data and the results are presented. This validation procedure showed that the nonlinear two-track vehicle dynamic model was most suitable for planning safe trajectories in critical traffic-scenarios. Finally, the self-developed simulation environment for the purpose of simulation of critical traffic-scenarios and data generation was presented briefly.

Chapter 3

Sampling-Based Trajectory Planning Algorithms

This chapter introduces two model-based trajectory planning algorithms that plan a safe trajectory in critical traffic-scenarios with multiple static and dynamic objects. These algorithms are modified versions of already proposed algorithm CL-RRT, a path planning algorithm of the MIT-team in the DARPA challenge in 2007. Therefore, these algorithms are named as the Augmented CL-RRT and Augmented CL-RRT+. These two variants will be compared with each other for many critical traffic-scenarios in the simulation environment.

The overview of this chapter is as follows: Section 3.1 describes the primary types of trajectory planning algorithms. It also presents a literature survey for the Rapidly-exploring Random Tree trajectory planning algorithm as this algorithm is the basis of the proposed trajectory planning algorithms in this work. Section 3.2 defines the specific problem for the trajectory planning algorithm in critical traffic scenarios with multiple static and dynamic objects. The Time-To-Collision criteria for the detection of the critical traffic-scenarios is described in Section 3.3. Section 3.4 provides a brief description of the CL-RRT algorithm, while Section 3.5 and 3.6 illustrates the details of the proposed Augmented CL-RRT and Augmented CL-RRT+ algorithm. Finally, Section 3.7 summarizes and compares the simulation results for critical traffic-scenarios with the Augmented CL-RRT and Augmented CL-RRT+ algorithm.

3.1 Background

The goal of safe trajectory planning is to produce a continuous set of actions for a robot, which is a vehicle in this work, for the time interval $[t_0, t_0 + \tau]$ from the the start state $\boldsymbol{s}_{init} \subset S$ to goal state $\boldsymbol{s}_{goal} \in S$ in state-space S avoiding collisions with obstacles defined by obstacle region $S_{obs} \in S$. Apart from collision avoidance constraints, nonholonomic con-

straints, which arise in the context of the vehicle trajectory planning, need to be considered. These nonholonomic constraints are expressed in the form

$$\dot{\boldsymbol{s}} = \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{u}), \tag{3.1}$$

in which $\boldsymbol{u}$ is the control input to the robot. The vector $\dot{\boldsymbol{s}}$ denotes the derivative of the robot state with respect to the time. The representation in Eq. 3.1 encodes a dynamical model. Many trajectory planning algorithms have been proposed in the robotics community. They are broadly classified into the following categories: grid-based algorithms, artificial potential field based Algorithms, reward-based algorithms, and sampling-based algorithms. These are described along with their advantages and disadvantages in the following sections.

3.1.1 Grid-Based Algorithms

In grid-based approaches, a grid is overlaid in the robot surrounding to enable discretization of the state-space and action-space. The most popular algorithms in this category for static environments are A* [HNR68] and its variants [KLF05, SD13]. D* [Ste94] and its variants [KL02, FS06] are grid-based algorithms for trajectory planning in dynamic environments. These algorithms aim to find a path from a starting state $\boldsymbol{s}_{init}$ to the goal state $\boldsymbol{s}_{goal}$ having the least cost. A cost function is typically defined for the behaviour of the robot in particular situations. A graph is generated starting from $\boldsymbol{s}_{init}$. This graph extends one of its edge in every iteration towards an adjacent cell. The decision towards which cell the path is to be extended, is taken based on two aspects. First, the cost of the already planned path from $\boldsymbol{s}_{init}$ to that cell. Second, an estimate of the cost required to extend the path to the goal state $\boldsymbol{s}_{goal}$. Specifically, the path is selected that minimizes

$$f(n) = g(n) + h(n), \tag{3.2}$$

where n is the cell towards which the path is to be extended, $g(n)$ is the cost of the path from the starting cell to cell n and $h(n)$ is a heuristic function that estimates the cost of the cheapest path from cell n to $\boldsymbol{s}_{goal}$. The heuristic function $h(n)$ is problem-specific. Algorithms terminate when the path it chooses to extend is a path from $\boldsymbol{s}_{init}$ to $\boldsymbol{s}_{goal}$ or if there are no paths eligible to be extended. D* and its variants quickly replan paths with a movement of an object in the surrounding of the robot. These algorithms suitable for trajectory planning in an environment with dynamic objects.

Although these algorithms have shown positive results, they need a discretized state-space and a discretized action-space. Therefore, they have limited suitability to vehicle trajectory planning. To overcome this limitation the resolution of the grids can be made very fine. However, the number of cells in grid grows exponentially as the dimension of state-space grows, which in turn also increases the computation time for finding a safe trajectory. This makes grid-based algorithms inappropriate for planning trajectories in multidimensional state-spaces.

3.1.2 Artificial Potential Field Based algorithms

The artificial potential field method [Kha85] assumes the robot moving in an artificial force field. The total potential U has two components: attractive potential U_{att} and repulsive potential U_{rep}. The goal position produces an attractive force which makes the robot move towards it. This potential can be modelled to suit a certain application or requirement. The most commonly used attractive potential U_{att} is

$$\mathsf{U}_{att}(\boldsymbol{s}) = \frac{1}{2}\xi_a d(\boldsymbol{s}, \boldsymbol{s}_{goal})^2, \tag{3.3}$$

where $d(\boldsymbol{s}, \boldsymbol{s}_{goal})$ is the distance between the robot position $\boldsymbol{s}$ and the goal position $\boldsymbol{s}_{goal}$ and ξ_a is the positive scaling factor for the attractive potential. The attractive force F, defined as the negative gradient of the attractive potential, becomes

$$F_{att}(\boldsymbol{s}) = -\nabla \mathsf{U}_{att}(\boldsymbol{s}) = \xi_a d(\boldsymbol{s}, \boldsymbol{s}_{goal}). \tag{3.4}$$

Thus, the attractive force decreases towards zero as the robot moves towards the goal.

In order to avoid the collision with an obstacles while the robot is moving towards the goal, a repulsive potential U_{rep} is generated, which is inversely proportional to the distance from the robot to obstacles such that the repulsive force is pointing away from obstacles. An example of the repulsive force is

$$\mathsf{U}_{rep}(\boldsymbol{s}) = \begin{cases} \frac{1}{2}\xi_r \left(\frac{1}{d(\boldsymbol{s},\boldsymbol{s}_{obs})} - \frac{1}{d_0}\right), & \text{if } d(\boldsymbol{s}, \boldsymbol{s}_{obs}) <= d_0 \\ 0, & \text{otherwise,} \end{cases} \tag{3.5}$$

where $d(\boldsymbol{s}, \boldsymbol{s}_{obs})$ is the shortest distance between the robot and the obstacle, d_0 is largest distance from the obstacle where it generates the negative potential, ξ_r is the positive scaling factor for the repulsive potential. The negative gradient of the repulsive potential, i. e., the negative force is

$$\begin{aligned} F_{rep} &= -\nabla \mathsf{U}_{rep}(\boldsymbol{s}) \\ &= \begin{cases} \xi_r(\frac{1}{d(\boldsymbol{s},\boldsymbol{s}_{obs})} - \frac{1}{d_0})\frac{1}{d^2(\boldsymbol{s},\boldsymbol{s}_{obs})}\nabla d(\boldsymbol{s}, \boldsymbol{s}_{obs}), & \text{if } d(\boldsymbol{s}, \boldsymbol{s}_{obs}) <= d_0 \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{3.6}$$

The total force applied to the robot is the addition of attractive and repulsive force and it determines the motion of the robot. The path planned with this resulting force is not necessarily a drivable path for the vehicle as no vehicle constraints are considered. Also, when the attractive force is equal and opposite of the repulsive force, the resultant potential force acting on the robot is zero and it gets trapped in local minima or oscillations. Also, it is challenging to plan paths in narrow spaces using potential field methods. It is because the cumulative repulsive force of two objects becomes very high. Even if it plans a path, the method is prone to oscillations [KB91]. Planning a trajectory through narrow spaces is vital for vehicle trajectory planning, especially in the critical traffic-scenarios with many

static and dynamic objects. There are some potential field approaches for vehicle trajectory planning [WM03, SPK11, RHN16] proposed in the literature, but they are limited to either static environments or environments with a single dynamic obstacle. Therefore, artificial potential field methods are not further considered for this work.

3.1.3 Reward-Based algorithms

Reward-based algorithms are based on choosing the best action in each state of the robot over the entire planning duration to get the maximum reward. These algorithms assume that the robot receives a positive or negative reward based on the action it takes. The reward function is defined such that the robot obtains the desired behaviour. A simple reward function for safe trajectory planning will provide a positive reward after reaching the goal while it will give a negative reward for colliding with other objects. The robot aims to choose a series of actions such that it receives the highest cumulative reward. These algorithms use the mathematical framework of the Markov decision process, which forms the basis of reinforcement learning as well.

These methods work better with discrete state-space and action-spaces. As the state-space or action-space grows, more complex models are needed. The combination of reinforcement learning with deep learning approaches, namely deep Q-learning, has been successful in achieving good results. A well-known example of this is Alpha-Go, developed by Deepmind, which beat the Go world champion. The point to note here is that Go has a huge state- and action-space, but they are still discrete. For a vehicle trajectory planning algorithm both are continuous. Go is only a two-player problem with each player playing in turns and the player wins by defeating the other player. Also, there is no constraint on the available computational resources and the real-time capability is not a requirement. On the other hand, the vehicle trajectory planning problem consists of many participants moving simultaneously and it has to be solved in real-time with limited available computational power. Thus, also this approach will not be considered further in this work.

3.1.4 Sampling-Based algorithms

These algorithms sample random robot states in the robot surrounding and retain only those states which are in the free space, i. e., in the space which is not occupied by other objects. The only two deterministic states are $\boldsymbol{s}_{init}$ and $\boldsymbol{s}_{goal}$. These algorithms construct a roadmap between these states. If the road connecting any two of the sampled random states is collision-free, then it is added to the roadmap. If a path in the roadmap connects the initial robot state $\boldsymbol{s}_{init}$ and final robot state $\boldsymbol{s}_{goal}$, then the planner is successful.

Many sampling-based methods for path planning are *probabilistically complete*, i. e., they will always find the collision-free path, if it exists, given infinite time. However, if it does not find a path, the reason for it is not definitive. It means either there is no possible collision-free path or the algorithm did not sample enough random robot states.

Another advantage of these algorithms is that they work well in high-dimensional state-spaces because unlike other algorithms, their run-time is not exponentially dependent on the dimension of the state-space. There are two popular sampling-based path planning algorithms: Probabilistic Roadmaps Planner and Rapidly-exploring Random Tree.

Probabilistic Roadmap Planner

The *Probabilistic Roadmap Planner* (PRM) is a type of sampling-based path planning algorithm. It consists of two phases: a construction and a query phase. These two phases are shown in the Fig. 3.1. In the construction phase, a roadmap is built. First, many random robot states are sampled and only the robot states which lie in free space are retained and others are ignored. Then, each random state is connected to some neighbors, typically either the k-nearest neighbors or all neighbors less than some predetermined distance. Among these connections, only the connections which are not intersecting other objects are retained. Finally, the robot states and connections are added to the graph until the roadmap is dense enough. In the query phase, the $\boldsymbol{s}_{init}$ and $\boldsymbol{s}_{goal}$ states to the nearest robot state are connected in the graph and an optimal path by a graph search algorithm such as the Dijkstra's algorithm [Dij59] is found. The proof of probabilistic completeness of the probabilistic algorithm is shown in [HLK06].

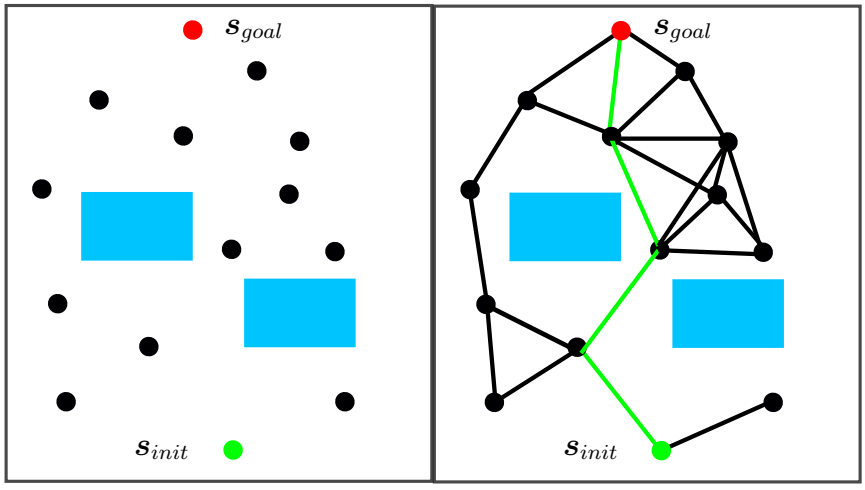


Figure 3.1: Construction and Query Phase of the Probabilistic Roadmap Planner.

Probabilistic roadmaps require many local connections to be made between the sampled robot states, therefore it is difficult to use them for robots like vehicles with nonholonomic constraints. Thus, they will not be considered further in this work.

Rapidly-exploring Random Tree

A Rapidly-exploring Random Tree (RRT) algorithm is designed for efficiently searching nonconvex high-dimensional spaces [Lav98] that have both collision constraints and differential constraints. It is an iterative sampling algorithm in which a tree of robot states is

constructed by incremental extension of the tree $\mathcal{T}$ using randomly drawn samples. The key idea behind this algorithm is to bias the exploration towards the unexplored parts of the space by sampling points in the state space and incrementally pulling the tree towards them.

The basic steps in the tree $\mathcal{T}$ construction, for a given robot initial state $\boldsymbol{s}_{init}$, with K iterations are shown in Algorithm 3.1. These steps are visualised in the Fig. 3.2. Initially, $\boldsymbol{s}_{init}$ is added as the first state to the tree $\mathcal{T}$. A random state, $\boldsymbol{s}_{rand} \in S$, is selected in each iteration. Step 4 finds the closest state $\boldsymbol{s}_{near}$ to $\boldsymbol{s}_{rand}$ which was added to the $\mathcal{T}$ in earlier iterations. If it is the first iteration, then the tree $\mathcal{T}$ has only the state $\boldsymbol{s}_{init}$. Therefore, the nearest state in the first iteration is always $\boldsymbol{s}_{init}$. In subsequent iterations, it uses a predefined distance metric to find the nearest neighbour from multiple states in the tree $\mathcal{T}$. Typically, the Euclidean distance is used as the distance metric. An input $\boldsymbol{u}$ is selected to extend the tree $\mathcal{T}$ towards $\boldsymbol{s}_{rand}$ from $\boldsymbol{s}_{near}$ for a predefined period of Δt to get the new state $\boldsymbol{s}_{new}$. Here, the extension is performed using the holonomic or nonholonomic constraints, such as in Eq. 3.1. Also other constraints like collision avoidance can be included in the algorithm . Hence, for the new state $\boldsymbol{s}_{new}$, the edge connecting the $\boldsymbol{s}_{new}$ and $\boldsymbol{s}_{near}$ is checked for collisions and it returns success upon no collisions and then this new state is added to the tree $\mathcal{T}$. This procedure is continued till either the number of samples exceeds a predefined threshold or one of the states found in K iteration is $\boldsymbol{s}_{goal}$. While adding the robot state $\boldsymbol{s}_{new}$ to the tree $\mathcal{T}$, other information such as the state $\boldsymbol{s}_{near}$ and the corresponding input $\boldsymbol{u}$ of the robot required to move from $\boldsymbol{s}_{near}$ to $\boldsymbol{s}_{new}$ is also stored, to trace back the trajectory from $\boldsymbol{s}_{goal}$, once it is found.

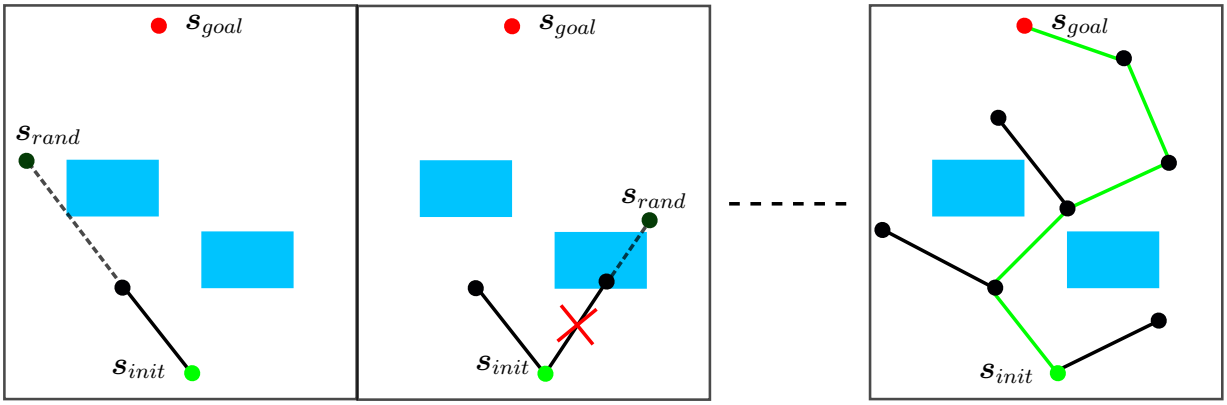


Figure 3.2: Steps in Constructing $\mathcal{T}$.

Additionally to having a simple way of constructing the $\mathcal{T}$, the RRT algorithm also has a desirable property of being biased towards places not yet visited. An RRT algorithm can be considered a Monte-Carlo way of biasing search into the largest Voronoi regions. At each iteration, the probability of a state, being selected for the extension, is proportional to the volume of its Voronoi region. Hence, the search is biased toward those nodes with the largest Voronoi regions. It enables the rapid exploration with the RRT algorithm.

Algorithm 3.1 `Generate_RRT`($\boldsymbol{s}_{init}$)

1: $\mathcal{T}.init(\boldsymbol{s}_{init})$;
2: **for** $k = 0$ to K **do**
3: $\quad \boldsymbol{s}_{rand} \leftarrow$ RANDOM_STATE();
4: $\quad \boldsymbol{s}_{near} \leftarrow$ NEAREST_NEIGHBOUR($\mathcal{T}, \boldsymbol{s}_{rand}$);
5: $\quad \boldsymbol{u} \leftarrow$ SELECT_INPUT($\boldsymbol{s}_{near}, \boldsymbol{s}_{rand}$);
6: $\quad \boldsymbol{s}_{new}, success \leftarrow$ NEW_STATE($\boldsymbol{s}_{near}, u, \Delta t$);
7: $\quad$ **if** *success* **then**
8: $\quad\quad \mathcal{T}.add\_vertex(\boldsymbol{s}_{new})$;
9: $\quad\quad \mathcal{T}.add\_edge(\boldsymbol{s}_{new}, \boldsymbol{s}_{near}, \boldsymbol{u})$;
10: $\quad$ **end if**
11: **end for**
12: RETURN $\mathcal{T}$;

Even with the uniform distribution sampling strategy, the RRT algorithm will be able to find a safe trajectory with nonholonomic constraints, if it exists, at the cost of a slow convergence rate. This is, of course, given that there is no restriction on the number of iterations. This property of RRT is called as *probabilistic completeness*. The proof for probabilistic completeness of RRT is given in [KSL+19].

The other hyperparameters in the RRT algorithm are the distance metric and the time interval Δt for the tree extension in one iteration. These two parameters define the edge between two states of $\mathcal{T}$. Typical values for these parameters are challenging to find and are dependent on the application. Generally, smaller edges provide more flexibility to find very complicated trajectories even in state-spaces with nonconvex obstacle regions, but then RRT algorithm suffers from high convergence time. On the other hand, the RRT algorithm with larger edges might not even be able to find a safe trajectory in cases where the RRT algorithm could have found a safe trajectory with small edges. Therefore, domain knowledge is vital to decide the length of edges in the RRT algorithm.

In comparison to PRM, RRT is always able to maintain a connected structure irrespective of the number of states. PRM suffers in performance because many extra edges are generated in attempts to form a connected roadmap. RRT works with single nearest neighbour queries, while PRM requires more expensive k-nearest neighbour queries. This leads to a lower number of collision checks, which are one of the most expensive operations in trajectory planning. With all these desirable properties, the RRT algorithm is a very attractive algorithm for vehicle trajectory planning in critical traffic-scenarios.

3.1.5 Survey of RRT Algorithm Variants

There is vast literature about RRT algorithms as many variants of this algorithm have been proposed. These approaches are divided into separate categories based on the different settings (e. g. sampling strategy) in the RRT algorithm. Usually, a variant of the RRT

algorithm combines different settings. Therefore they are classified based on the primary setting change, which is significantly different from other RRT algorithm variants.

Sampling Strategy

The most common criteria for creating RRT variants is defining a new sampling strategy. The basic RRT algoritfhm was proposed with uniform sampling schemes. This is a drawback, especially in scenarios having narrow free spaces as the probability of sampling in narrow spaces is also low. Also, because of the randomness in its nature, the convergence time for the RRT algorithm with uniform sampling can be high with comparison to deterministic planning algorithms even in scenarios with wide free spaces. Therefore, many sampling strategies have been suggested to overcome these shortcomings. In [WAS99, HK00], sampling along the medial axis of free space is suggested for finding an optimal path such that the samples have maximum clearance from S_{obs}. On the other hand, [AW96] proposes sampling on S_{obs} to find the paths in narrow passages quickly.Heuristically Guided RRT [US03] includes heuristics in the RRT algorithm expansion step so that randomly sampled states are probabilistically added in proportion to their heuristic value to the tree. The goal biasing, in which a goal state is deterministically sampled in fixed intervals, is introduced in [KL00]. A significant reduction in planning time is achieved in [Yan13, KKY+16] by increasing the density of sampling around the goal region once the tree approaches it. In [KLY14], a methodology is presented for defining sampling clouds for allocating promising samples and refining them after the initial solution is found. Fast Marching Tree [JP13] uses a marching method to search a batch of samples in order of increasing cost-to-come, similar to Dijkstra's algorithm [Dij59].

Heuristic

In this section, methods that refine the trajectory found by an RRT algorithm to achieve lower cost/planning time are summarized. RRT*, an asymptotically optimal variant of RRT, is proposed in [KF11]. This algorithm improves the path within the given planning time. However, the algorithm provides no guarantees on reaching an optimal solution under certain criteria and time constraints. This property is known as asymptotic optimality. In literature, many other variants have been proposed to reduce the computation time for finding an optimal solution.

Karaman et al. [KWP+11] and Akgun and Stilman [AS11] both use heuristics to accelerate the convergence of RRT*. Karaman et al. [KWP+11] remove states whose current cost-to-come to a particular state plus a heuristic estimate of cost-to-go to the goal from the same state is higher than the current solution. Akgun and Stilman [AS11] reject samples that are heuristically estimated to be unable to provide a better solution. Kiesel et al. [KBR12] first calculated a heuristic by solving a coarse discretization of the planning problem with Dijkstra's algorithm, which is then used to bias RRT* sampling to the regions of the problem domain where solutions were found. A similar approach is used in [BBS13] using the A* algorithm. RRT# [AT13] uses heuristics in dynamic environments to limit the tree

to regions of the problem domain that can improve an existing solution. Anytime RRT deals with a lack of computational time for path improvement by generating an initial suboptimal solution [FS06] and then improving the solution in the remaining time. Waypoint caches are also used to guide replanning with anytime RRT [ZWWW09]. The proposed planner in [Von18] first solves a simplified version of the problem that is achieved, e.g., by reducing the geometry of the robot. This approximate solution is then used to guide the search in the state-space for a less relaxed version of the problem. All approaches, mentioned in this section, avoid unnecessary computational effort but do not influence the initial search as heuristics are not applied until a solution is found. This means these algorithms require at least an initial approximate solution to apply heuristics.

Replanning

As S_{obs} changes, especially in dynamic environments, the planned paths may become invalid. In such situations, a new path has to be planned. The information gathered from the path planning in past can be used to replan the path efficiently. ERRT [BV03] maintains the location of the discarded states as well as waypoint cache, and bias the search based on this information. It is motivated by the assumption that if the algorithm is updated at a high frequency, only a small percentage of the original tree needs to be modified. Planning a new trajectory from scratch every time is very expensive. Therefore, many variants of the RRT algorithm have been proposed to replan the path by pruning or updating the tree. Dynamic RRT [FKS06] builds on the idea that it is more efficient to repair the existing tree than to rebuild an entirely new one. Unlike ERRT, only the colliding configurations and their child nodes are discarded in an efficient manner. Reconfigurable Random Forests (RRF) provided a framework for managing trees by discarding states in changed S_{obs} and colliding paths. This leads to the emergence of separate trees, which then planner attempts to reconnect. Multipartite RRT [ZKB07] combines the strategy of biasing the search towards discarded configurations, similar to ERRT. It also rebuilds the tree, like Dynamic RRT and maintains separate detached forests, like RRF. RRT$^{\mathrm{X}}$ [OF16] is a motion replanning algorithm for real-time navigation through a dynamic environment. After finding the shortest path for a specific configuration, the RRT$^{\mathrm{X}}$ algorithm replans the path to a goal by continually repairing it as changes to the state-space are detected. However, the first shortest path is found offline and finding this path by taking into account the predicted motion of objects is not the focus of RRT$^{\mathrm{X}}$.

Local Planning and Smoothing

Smoothing techniques rely on using a curve to interpolate or fit the given waypoints. Dubin's path [BBSL94] is commonly used for nonholonomic vehicles that are bound by a minimum turning radius [KFT^{+}08]. They combine circular arcs and straight lines to generate optimal paths. However, the curvature of the path may not be continuous. Curvature continuous paths were proposed using Clothoids in [KH89]. Clothoids have no closed-form solution and thus provide computational challenges to synthesize in real-time [MW04].

Bezier curves were proposed for smoothing [YS10] and were used for local planning in SRRT [YGS13, Yan13]. In [NMC10], seventh order Bezier curves are used to find smooth trajectories that do not violate the kinematic constraints of the vehicle. B-spline interpolation was used to generate smooth trajectories for an RRT planner in dynamic driving scenarios [MBS06a, SDSS09, GYZ+17]. The local modification support was exploited by generating a feasible path and then subsequent local adjustments are performed to ensure dynamic feasibility. The main advantage of using splines is that the kinodynamic planning is limited to a lower-dimensional space, thus making the planning in real-time possible.

Multiple Trees

Multiple RRTs can also be constructed simultaneously and connected to each other for finding collision-free paths. In this section, RRT variants generating multiple trees are mentioned. RRT-Connect [KL2000] uses two trees to perform a bidirectional search. One tree is rooted at the start, whereas the other is at the goal. The search is complete when the two trees are connected. In [ESJ16], B-spline parameterized curves are used for bidirectional search. Triple RRT [WXL+10] and Multiple RRT [ZKB07, HWL17] generates two trees from start and goal configurations and more trees from a narrow region which is identified using the bridge test. A problem arises when attempting to connect two trees for differentially constrained systems where the local planning is not a simple straight line.

Machine Learning

In this section, the applications of machine learning algorithms, for making the subtasks of the RRT algorithms more efficient, are described. NoD-RRT [LCLX18] uses a neural network to predict the cost between two given states considering nonlinear constraints for the tree expansion. An approach for efficient sampling-based on learning a Q-function to avoid obstacles is presented in [HL18]. In [PHCM18] a fully convolutional neural network predicts the paths to be learned from expert demonstrations, which is further refined with RRT* algorithm. Randomized statistical path planning [DK07] shows a formulation to extract motion primitives from training data and outlines possibilities of how machine learning can be used to learn heuristics. DeepSMP [QY18] encodes the raw point cloud data using a contractive autoencoder and uses this encoding as an input to another neural network along with the start and goal robot states to generate feasible samples for computing collision-free paths with the RRT algorithm. A learned Gaussian Mixture Model distribution is used for the biased-sampling in learned free spaces [HL16] to decrease the number of collision checks drastically for the trajectory planning with the RRT algorithm. Biased sampling in free spaces might increase the probability of finding collision-free states, but it is not necessarily a good sampling strategy for the long term trajectory planning, which requires a long chain of collision-free states. sRapidly Exploring Learning Trees [SMW17] learns the cost functions of Optimal Rapidly Exploring Random Trees (RRT*) from demonstration, thereby making inverse learning methods applicable to more complex tasks. A nonlinear parametric model is trained to learn the distance metric [PA15] with

nonlinear constraints between states of RRT essential to choose the nearest state of the sample.

Kinodynamic Planning

Kinodynamic planning deals with the kinematic, nonholonomic or dynamic constraints imposed on the robotic system such as a vehicle. The previously presented planners were purely geometric, considering only the feasibility of the path. In comparison to the sampling-based planners like the RRT algorithm, deterministic planners usually suffer from high computational cost with kinodynamic constraints.

In some variants of the RRT algorithm, path planning and kinodynamic constraints are decoupled as explained in the Section 3.1.5 where planners generate a path that relaxes all kinodynamic constraints followed by trajectory modification to obey the constraints.

Closed-Loop RRT (CL-RRT) [KTF+09] runs a forward simulation using a vehicle model to compute the predicted trajectory whose feasibility is checked against the vehicle and environmental constraints. It uses a drivability map, which is regularly updated to check the validity of nodes and edges of the tree. The algorithm takes into account the vehicle dynamics by using a controller. It considers a complete stop as a safe state at the end of a trajectory. In critical traffic situations with many dynamic obstacles, this might not be the right choice since zero velocity does not necessarily imply a safe state. CL-RRT$^{\#}$ leverages idea from RRT$^{\#}$ to combine with CL-RRT to generate better quality trajectories using closed-loop predictions with dynamic constraints. [KTF+09] presents a methodology for finding an informed subset of samples for differential constraints to improve the efficiency of the RRT algorithm.

There are several issues about kinodynamic planning. It is inherently a high dimensional problem and the state equation of the system must be known. Discarding kinodynamic constraints during planning may lead to highly suboptimal solutions that involve complicated manoeuvres. In worst cases, the robot may not be able to execute the planned paths, resulting in unrecoverable situations that lead to a collision. For some systems, attempting to accurately model all the effects overcomplicates the model and increase the planning space dimensions and the search complexity.

Planning in a state-space that has narrow corridors is one of the challenges in the planning path problem with an RRT algorithm. Kinodynamic constraints limit the motion of the robot, essentially creating narrow passages in the state-space. The state-space is, traditionally, defined into free and obstacle state-space. The free space becomes narrower with kinodynamic constraints if states those are unreachable and states from which a collision is unavoidable are removed from free space. High dimensional planning, combined with narrow free passages, leads to a significant increase in the convergence time of the RRT algorithm. Therefore, kinodynamic planning has been limited to simulation-based planning applications. It is observed that the planning time can reach several minutes in some simulation scenarios [LJJK01].

Limitations of RRT variants

A huge number of RRT algorithm variants have been proposed for solving different robotics problems successfully, yet the applications of RRT algorithms have not been researched, especially in vehicle trajectory planning in critical, complex traffic-scenarios with many static and dynamic road traffic-participants. The possibility of consideration of vehicle dynamic constraints in the path planning task and the need of comparatively low computational resources for planning trajectories in a high dimensional state-space makes the RRT algorithm very attractive for the vehicle trajectory planning application. However, it is still not used for the vehicle safe trajectory planning because of the several disadvantages associated with it.

The most crucial thing in vehicle trajectory planning, especially in a critical traffic-scenario, is the safety than the comfort of the vehicle occupants. The RRT algorithm is a probabilistically complete algorithm, but it cannot guarantee to find a safe path in real-time. For critical-traffic scenarios with many static and dynamic objects, the need for computational resources is even higher. This is because the trajectories need to be planned in narrow spaces, which is difficult with the RRT algorithm, as explained in the previous section. Also, the trajectories need to be planned with simultaneous interventions in the lateral and longitudinal dynamics. This increases the action-space for the RRT algorithm, and thus the computation time. Very few variants of the RRT algorithm [MBS06b, OF16, KFT+08] have been proposed which use simultaneous intervention in lateral and longitudinal dynamics of the vehicle, but they either require controllers with high computing power or a lot of precomputation.

Equipping vehicles with controllers having high computing powers such as GPU is only possible for high-end vehicles. Even if the vehicle has onboard controllers with high computing power, it cannot be guaranteed to find the collision-free path with the RRT algorithm in the provided time. The vehicle may have reached a state from which the collision cannot be avoided. There is no simple fail-safe state for a vehicle, such as in many other robotics applications, where a complete stop is a fail-safe state. No RRT algorithm variant suggests what the vehicle should do in such situations.

There are various heuristics and sampling strategies that have been proposed for decreasing the computation time for path planning with an RRT algorithm, but they are very scenario-specific and require at least an initial approximate solution. A small change in the scenario can make those heuristic or sampling strategies ineffective, sometimes making the problem more difficult.

Another challenging aspect with planning paths with multiple dynamic road-participants are paths that look safe at one instance may not be in a future instant. Some RRT algorithm variants replan the path by pruning and rewiring the RRT. This is not a good idea, especially in safety-critical applications like vehicle trajectory planning. The safe path should be planned by considering the future behaviour of other traffic-participants. Otherwise, the vehicle may enter into a situation from which it is not possible to escape.

All of these drawbacks of the RRT algorithm, particular requirements of vehicle trajectory planning and the unavailability of a suitable RRT algorithm variant or, in general, a trajectory planning algorithm, necessitate the extension of the RRT algorithm.

3.2 Problem Formulation for Vehicle Safe Trajectory Planning

The primary objective of this work is to plan a *safe trajectory* for the EGO vehicle in critical traffic-scenarios with multiple static and dynamic objects. Generally, a safe trajectory means finding a collision-free trajectory. However, a collision may be unavoidable in a particular traffic-scenario, even with the combination of lateral and longitudinal dynamics interventions. This is especially possible in critical traffic-scenarios, where many dynamic objects create narrow free spaces. Sometimes, the trajectory planning algorithms fail to find a collision-free trajectory because of insufficient computational resources. In any of these cases, a trajectory should be planned such that it mitigates the severity of injury. Therefore, this work extends the definition of finding a safe trajectory as either finding a collision-free trajectory or a trajectory with predicted low severity of injury if a collision-free trajectory is not found.

The aim of the trajectory planning algorithm in this work is an autonomous intervention in only critical traffic-scenarios. Critical traffic-scenarios are defined as scenarios in which a collision is predicted for the EGO vehicle in the time interval $[t_0, t_0 + t_p]$ and it cannot be avoided by full braking. The exclusion of traffic-scenarios where a collision can be avoided by full braking is because of the presence of the Autonomous Emergency Braking system in modern vehicles. Therefore, the focus of the trajectory planning problem is on traffic-scenarios, where simultaneous intervention in the lateral and longitudinal dynamics of the vehicle is necessary for the collision avoidance/mitigation.

The task of safe trajectory planning does not end by planning a collision-free trajectory for a specified time interval. The trajectory planning algorithm can bring the EGO vehicle from one critical state to another critical state. For example, the EGO vehicle may avoid a collision by steering and changing the lane, but it may lead to a vehicle state that can hardly be controlled after the steering manoeuvre. Therefore, the end position of the planned trajectory also has to be evaluated. The task of the trajectory planner should be to plan a trajectory from one critical state to a safe state from which the driver or a software module in autonomous driving software for comfortable driving can take control of the vehicle easily. Therefore, a trajectory should be chosen such that it is collision-free for a longer time interval than the one for which the trajectory is planned.

In this work, a two-dimensional space is assumed for modelling a robot state in a traffic-scenario. The vehicle has nonlinear dynamics as in the Eq. 3.1

$$\dot{\boldsymbol{s}}(t) = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{u}(t)), \tag{3.7}$$

where $\boldsymbol{u}(t) \in \mathbb{R}^m$ is the m-dimensional control input at time t and $\boldsymbol{s}(t) \subset \mathbb{R}^2$ represents the set of points occupied by the vehicle at time instance t. The initial state at t_0 is given by $\boldsymbol{s}(t_0)$. The path-planning problem implies the design of the control input $\boldsymbol{u}(t)$ over a finite prediction time-horizon of length τ_1, i. e., $t \in [t_0, t_0 + \tau_1]$. The set of constraints like bounds on the control input, static and dynamic obstacle avoidance or rules imposed by the road must be taken into account when computing $\boldsymbol{u}(t)$, so that the resulting $\boldsymbol{s}(t)$ is collision-free, i. e., $\boldsymbol{s}(t) \in S_{free}(t)$, where $S_{free}(t)$ expresses the road area in $\mathbb{R}^2$ which is not occupied by other objects at prediction-time t. To plan drivable trajectories, a nonlinear two-track model, as presented in Chapter 2, is used as the dynamic constraint function $\boldsymbol{f}$ in the Eq. 3.7.

A prediction time-horizon of approximately $\tau_1 \approx 2$ seconds is a suitable value for performing avoidance manoeuvre in most critical traffic-scenarios. In order to make sure that the trajectory is collision-free for a longer time interval than it is planned, a longer prediction time-interval $[t_0, t_0 + \tau]$ is considered. This prediction interval is broken down into two subintervals $[t_0, t_0 + \tau_1]$ and $[t_0 + \tau_1, t_0 + \tau]$. The collision-free trajectories are planned in the first subinterval and the ease of controlling the vehicle in the second subinterval without a collision towards the goal is taken as one of the measures for the final selection of the collision-free trajectory found in first subinterval.

Therefore, the problem of trajectory planning in critical traffic-scenarios is stated as follows. If a critical traffic situation is identified, find multiple control inputs $\boldsymbol{u}(t)$ so that resulting trajectories, which are sequences of states $\boldsymbol{s}(t)$, are collision-free, i. e., $\boldsymbol{s}(t) \in S_{free}(t)$ for $t \in [t_0, t_0 + \tau_1]$, and estimate the level of safety of the resulting trajectories in the interval $[t_0 + \tau_1, t_0 + \tau]$. If no collision-free trajectory is found, then estimate the level of safety from the predicted severity of injury along the found trajectories. Finally, choose the control input with the highest level of safety.

3.3 Detection of Critical Traffic Scenarios

As mentioned in Section 3.2, the proposed trajectory planning algorithms are activated only in the critical traffic-scenarios in which a collision cannot be avoided just by braking. Therefore, it needs to be detected if a traffic-scenario is critical in the first place. The Time-To-Collision (TTC) criticality criteria is used for defining the criticality of the traffic-scenario. If a collision of the EGO vehicle is detected with any other road traffic-participant within a TTC of two seconds in the future and it cannot be avoided by braking, then the scenario is said to be a critical traffic-scenario. Road traffic-participants include static objects like parked vehicles, trees, buildings or any other dynamic objects like other vehicles, pedestrians, bicycles, etc.

In order to detect a future collision for the EGO vehicle, the predictions of other road traffic-participants are necessary. There are numerous motion models proposed in the literature. They are classified into linear motion models and curvilinear motion models

[SRW08]. Linear motion models assume constant velocity or constant longitudinal acceleration. Although these models are simpler, they are only applicable on straight roads as they assume only straight motions and do not take rotations, especially yaw rate into account. Curvilinear motion models, on the other hand, take into account the rotations around the z-axis. They are further divided into types based on the state variables such as velocity, acceleration or steering angle. From a geometrical point of view, nearly all curvilinear models are assuming that the vehicle is moving on a circular trajectory.

The curvilinear motion models describe the motion of road vehicles accurately only for small prediction intervals ($\approx 0.5s$). It is because they consider only current values of vehicle parameters and not the details of the road infrastructure. Also, for pedestrians and bicyclists, suitable motion models need to be defined. Therefore, it is assumed in this work that vehicles tend to follow their lane with constant velocity and pedestrians travel linearly towards their current direction of travel with constant velocity. These are fair assumptions for the time interval of 4 seconds, which is the total time (τ) for which the trajectory is planned. Predictions with these assumptions generate the future positions $\boldsymbol{s}_{obj,n}(t)$ of obstacles in the vicinity of the EGO vehicle for the next τ seconds. A collision between the EGO vehicle and the n^{th} object at time t occurs if the indicator function

$$\mathrm{I}_{obj}(t) = \begin{cases} 1, & \text{if } \boldsymbol{s}(t) \bigcap S_{obj}(t) \neq \varnothing \\ 0, & \text{otherwise,} \end{cases} \tag{3.8}$$

or

$$\mathrm{I}_{road}(t) = \begin{cases} 1, & \text{if } \boldsymbol{s}(t) \bigcap S_{road}(t) \neq \varnothing \\ 0, & \text{otherwise,} \end{cases} \tag{3.9}$$

have the value 1 for any value of $t \in [t_0, t_0 + \tau]$. Fig. 3.3 shows further steps in the detection of a critical traffic-scenario. The difference between the time instance t_c, when a collision is identified, and the current time instance t_0 is the TTC. If the TTC is less than 2 seconds, and the collision cannot be avoided just by full braking, then the traffic-scenario is considered as a critical traffic-scenario. Here, it is assumed that the EGO vehicle is equipped with sensors that give the information about the current state of other obstacles like their position, velocity, acceleration, yaw angle, etc. and also the road infrastructure details like road width, curvature, etc.

3.4 Closed-Loop RRT

The trajectory planning algorithm needs to plan trajectories with simultaneous interventions in the lateral and longitudinal dynamics of the vehicle to avoid/mitigate the collisions in critical traffic-scenarios. As explained in Section 3.1.4, the RRT algorithm is suitable as it can plan accurate trajectories with complex vehicle dynamic models such as the nonlinear two-track model explained in Chapter 2. However, it suffers from lots of limitations, as described in Section 3.1.5. In order to overcome these limitations, different variants of

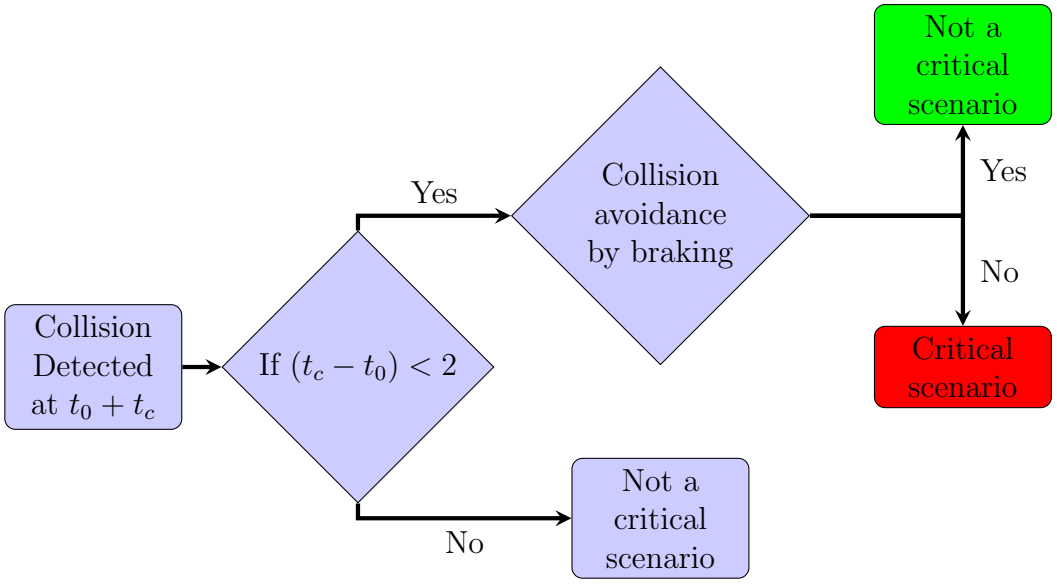


Figure 3.3: Detection of Critical Traffic-Scenario in this Work.

the RRT algorithm are proposed in this work. These variants are generated by introducing extensions to an already proposed variant of the RRT algorithm, namely closed-loop RRT (CL-RRT) [KTF+09]. It is a path planning algorithm developed by Team MIT in the 2007 DARPA Urban Challenge. The novelty of this algorithm lies in the use of closed-loop prediction in the framework of the RRT algorithm. Unlike the standard RRT algorithm, a reference input $\mathbf{r}$ to the controller is sampled. It is used to generate the input $\boldsymbol{u}$ that is used for the forward simulation using the vehicle model in the loop while extending the tree to find new robot states $\boldsymbol{s}_{new}$. A pure pursuit controller with simple PI controllers for calculating the steering and speed commands is used. The feasibility of the predicted trajectory is checked against obstacles and environmental constraints using the drivability map in which it maintains the environment occupancy information. The main advantages of the forward simulation are that it can easily incorporate any nonlinear control law and nonlinear vehicle dynamics, and the resulting trajectory is dynamically feasible. The flow diagram for the CL-RRT algorithm is shown in Fig. 3.4. This diagram shows how the tree is extended in one iteration to find the new state $\boldsymbol{s}_{new}$. The blue box shows the framework of the controller and the vehicle dynamic model, which uses the reference $\mathbf{r}$ at each time-step to extend the trajectory from $\boldsymbol{s}_{near}$ towards $\boldsymbol{s}_{rand}$. Finally, the found new state $\boldsymbol{s}_{new}$ and the edge from $\boldsymbol{s}_{near}$ to $\boldsymbol{s}_{new}$ is checked for the collisions using the drivability map.

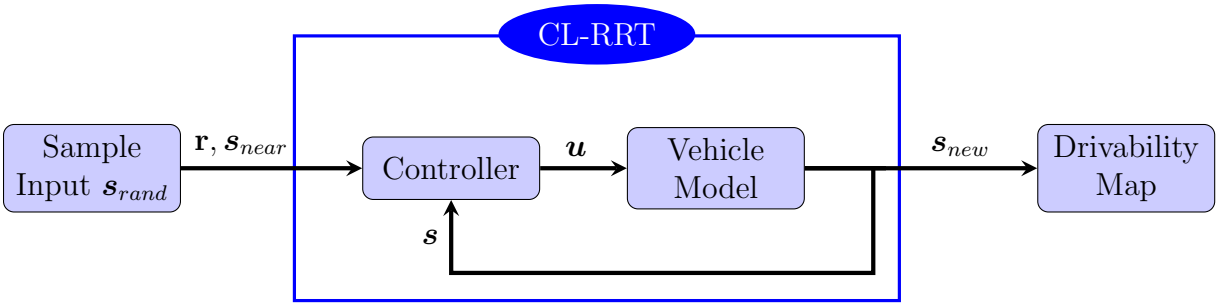


Figure 3.4: Flow Diagram for the Closed-Loop RRT Algorithm.

The reference input $\mathbf{r}$ is defined as a piecewise linear path for simplicity. The speed profile of the vehicle is always defined in the same way with three segments: initial ramp up, coasting, and ramp down, such that the vehicle comes to a stop at the end as this algorithm assumes that the stopped vehicle is safe. This is not true in critical traffic-scenarios especially with multiple dynamic objects which can collide with the stopped EGO vehicle. The limitations imposed on defining the reference path and speed profile also limit the possibilities of finding a safe trajectory in narrow free spaces of critical traffic-scenarios. Also, this algorithm uses the nonlinear bicycle model as vehicle model, which does not define the vehicle dynamics as detailed as a nonlinear two-track model. Therefore, the planned trajectory may not be followed accurately, especially in harsh manoeuvres.

In order to account for dynamic objects, the CL-RRT algorithm maintains a drivability map which is regularly updated. The validity of robot states in the CL-RRT is continuously checked against this drivability map and invalid robot states are discarded. In traffic-scenarios having multiple dynamic objects, this will lead to a frequent discarding of robot states. This happens since those states can become critical states in future time-steps. Often discarding the robot states in the tree is not efficient as computational resources used to find them are wasted.

Alg. 3.2 shows the pseudo-code for the CL-RRT algorithm. The changes in the CL-RRT algorithm with respect to the basic RRT algorithm are highlighted in the pseudo-code. It is clear from this code that the CL-RRT algorithm plans the trajectory from the start to the end of the vehicle manoeuvre at every time-step. Therefore, it continuously updates the states in the tree $\mathcal{T}$ at each time-step. The continuous planning or replanning of a collision-free trajectory is not the focus of this work. The aim is the autonomous intervention when the EGO vehicle encounters a critical traffic-scenario. It requires the ability to plan the trajectory entirely from scratch at a particular time-step with the limited computational resources available. It is also worth to mention that the team MIT used a Car-PC having high computational resources for running the CL-RRT algorithm.

Because of all the above reasons, the CL-RRT algorithm is not suitable for safe trajectory planning in critical traffic-scenarios. However, it contains a closed-loop controller and a vehicle dynamic model, which allows to plan drivable trajectories with nonholonomic constraints of the vehicle. Therefore, this work proposes new variants of the RRT algorithm

Algorithm 3.2 `CL_RRT`($\mathcal{T}$)

1: $DrivabilityMap, \mathcal{T} \leftarrow$ UPDATE_STATES($\mathcal{T}$);
2: **for** $k = 0$ to K **do**
3: $\boldsymbol{s}_{rand} \leftarrow$ RANDOM_STATE();
4: $\mathbf{r}, \boldsymbol{s}_{near} \leftarrow$ SAMPLE_INPUT($\mathcal{T}, \boldsymbol{s}_{rand}$);
5: $\boldsymbol{s}_{new} \leftarrow$ CONTROLLER ($\boldsymbol{s}_{near}, \mathbf{r}, \Delta t$);
6: $success \leftarrow$ CHECK_CONSTRAINS($DrivabilityMap$);
7: **if** $success$ **then**
8: $\mathcal{T}.add\_vertex(\boldsymbol{s}_{new})$;
9: $\mathcal{T}.add\_edge(\boldsymbol{s}_{new}, \boldsymbol{s}_{near}, \boldsymbol{u})$;
10: **end if**
11: **end for**
12: RETURN $\mathcal{T}$;

by extending the CL-RRT algorithm.

3.5 Augmented CL-RRT Algorithm

The Augmented CL-RRT (ARRT) algorithm is the first variant of the RRT algorithm developed by extending the CL-RRT algorithm. It uses the basic framework of the CL-RRT algorithm comprising the controller and vehicle dynamic model for the extension of the tree in every iteration. As described in Section 3.2, the safe trajectory planning problem is divided into two time intervals $[t_0, t_0 + \tau_1]$ and $[t_0 + \tau_1, t_0 + \tau]$. The first time interval is used for the planning of safe trajectories while the estimation of the level of safety of those trajectories is done in the second time interval. Apart from this, there are several extensions introduced in the ARRT algorithm to make it suitable for trajectory planning in critical traffic-scenarios. Some of these extensions are indicated in different colors in the Fig. 3.5. The corresponding algorithmic steps for this extension are marked in the same colors as in Alg. 3.3. These extensions are described in detail in the following sections.

3.5.1 Temporal Drivability Map $S_{free}(t)$

The CL-RRT algorithm plans the trajectory using the drivability map generated using the recent measurements from the environment. The drivability map is updated continuously as new measurements are received, which is again used to update the states in the tree. It does not anticipate the behaviour of other road traffic-participants for planning trajectories. This approach works well in static environments and, to some degree, in dynamic environments where objects are moving at very low velocity.

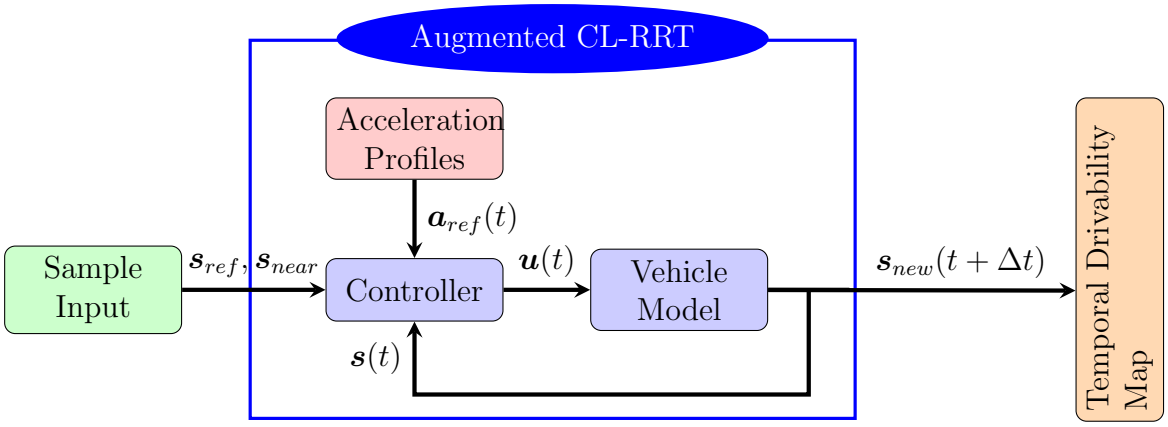


Figure 3.5: Flow Diagram for the Augmented CL-RRT Algorithm.

In critical-traffic scenarios with dynamic objects moving with high velocity, a safe trajectory needs to be planned using the predictions of other traffic-participants. This ensures that the planned trajectory is also safe in the future and the vehicle is not entering a false safe state which later becomes even more critical when the vehicle actually arrives in that state. The predictions of other road traffic-participants calculated for detecting the critical traffic-scenarios in Section 3.3 are also considered for planning safe trajectories. These predictions are used to construct a temporal drivability map $S_{free}(t)$, which defines the free space available for driving, at time instances $t \in [t_0, t_0 + \tau]$. Denoting the area S_{road} in $\mathbb{R}^2$ that is covered by the road, the free space $S_{free}(t)$ available at time t in an environment with total N_{obj} road traffic-participants is given as

$$\begin{aligned} S_{free}(t) &= S_{road} \setminus \bigcup_{n=1}^{N_{obj}} \boldsymbol{s}_{obj,n}(t), \\ &= S_{road} \setminus S_{obj}, \end{aligned} \tag{3.10}$$

where $\boldsymbol{s}_{obj,n}(t) \in S$ is the area occupied by the predictions of the n^{th} object at time instance t.

3.5.2 Augmented Robot States in the Tree $\mathcal{T}$

In the CL-RRT algorithm, the position of the vehicle defines the states in the tree. However, as described in the previous section, not just the position, but also the time at which the vehicle reaches that particular state is of vital importance in critical traffic-scenarios. If the EGO vehicle reaches these states, a little early or a little late, it might lead to a collision. Therefore, the robot states $\boldsymbol{s}_{new}$ found during the extension of the tree should be collision-free for the particular time instance at which the vehicle is predicted to reach there. Along with the robot states, the trajectory joining the state $\boldsymbol{s}_{new}$ to the tree should also be collision-free. The temporal drivability map is used for this collision checking. The

Algorithm 3.3 `Augmented CL_RRT`$(\boldsymbol{s}_{init}, \boldsymbol{v}_{init})$

$\boldsymbol{s}_{goal} \leftarrow$ GOAL_SELECTION $(S_{free}, \boldsymbol{v}_{init})$
for $n = 0$ to N **do**
  $\mathcal{T}_n.init(\boldsymbol{s}_{init})$;
  $\boldsymbol{a}_{x,n} \leftarrow$ ACCELERATION_PROFILE(n);
  for $k = 0$ to K **do**
    $\boldsymbol{s}_{rand} \leftarrow$ RANDOM_STATE();
    $\boldsymbol{s}_{ref}, \boldsymbol{s}_{near} \leftarrow$ SAMPLE_INPUT$(\mathcal{T}, \boldsymbol{s}_{rand})$;
    $\boldsymbol{a}_{ref}(t) \leftarrow$ REFERENCE_ACCELERATION$(\boldsymbol{a}_{x,n}, t)$;
    $\boldsymbol{s}_{new} \leftarrow$ FIND_STATE $(\boldsymbol{s}_{near}, \boldsymbol{s}_{ref}, a_{ref}(t), \Delta t)$;
    $success \leftarrow$ CHECK_CONSTRAINS$(\boldsymbol{s}_{free}(t))$;
    if $success$ **then**
      $\mathcal{T}_n.add\_vertex(\boldsymbol{s}_{new})$;
      $\mathcal{T}_n.add\_edge(\boldsymbol{s}_{new}, \boldsymbol{s}_{near}, \boldsymbol{u})$;
    else
      $severity \leftarrow$ CHECK_SEVERITY$(\boldsymbol{s}_{new}, \boldsymbol{s}_{obj},)$
      if $severityislow$ **then**
        $\mathcal{T}_n.add\_vertex(\boldsymbol{s}_{new})$;
        $\mathcal{T}_n.add\_edge(\boldsymbol{s}_{new}, \boldsymbol{s}_{near}, \boldsymbol{u})$;
        $\mathcal{T}_n.add\_severity(severity)$;
      end if
    end if
  end for
end for
$\mathcal{T}^* \leftarrow$ COMPARE_TREES$(\mathcal{T}_1, \ldots, \mathcal{T}_N)$
RETURN $\mathcal{T}^*$;

calculation of the time to reach a particular robot state is possible because of the use of vehicle dynamic models for the extension of the tree.

As the RRT algorithm is iterative, the tree is extended from robot states that were found in previous iterations. In order to do this using vehicle dynamic models, the parameters of the robot state from which the extension is carried out need to be known. These parameters include the physical parameters of the vehicle, including the velocity v, the steering angle δ, the yaw angle ψ as well the time t when that state was predicted to be reached. Therefore, this augmented information of each robot state is stored in the tree. Therefore, this algorithm is named as the Augmented CL-RRT (ARRT).

3.5.3 Controller Input and Output

The CL-RRT algorithm uses a piecewise linear path and a fixed piecewise linear acceleration profile as an input to the controller for simplicity. Therefore, the algorithm is also suitable for finding the safe trajectories only in simple traffic-scenarios with few objects moving at low velocities. In order to find a safe trajectory in complex, critical traffic-scenarios with multiple dynamic objects which might be moving with high velocities, the input to the controller needs to be adapted accordingly. As a nonlinear two-track vehicle dynamic model is used with the controller, it needs to calculate the input $\boldsymbol{u}(t)$ for the two-track vehicle dynamic model which comprises of four steering angles of the wheels with respect to the longitudinal axis of the vehicle $\delta_{fl}(t), \delta_{fr}(t), \delta_{rl}, (t), \delta_{rr}(t)$ and the four longitudinal slip values of the tires $s_{l,fl}(t), s_{l,fr}(t), s_{l,rl}(t), s_{l,rr}(t)$, where the letters in the subscript stand for "front-left", "front-right", "rear-left", and "rear-right". Thus, the output of the controller, i. e., the input to the vehicle dynamic model is

$$\boldsymbol{u}(t) = \{\delta_{fl}(t), \delta_{fr}(t), \delta_{rl}(t), \delta_{rr}(t), s_{l,fl}(t), s_{l,fr}(t), s_{l,rl}(t), s_{l,rr}(t)\}. \tag{3.11}$$

The steering wheel determines the angles in a vehicle, whereas the acceleration or brake pedal determine the longitudinal slip values. The angle of a tire with respect to the longitudinal axis of the vehicle is responsible for the lateral force acting at this tire and the longitudinal slip value is responsible for the longitudinal force acting at the tire. Therefore, some changes are proposed in the framework of the ARRT algorithm for both lateral and longitudinal dynamic interventions of the vehicle.

Steering Controller

In the CL-RRT algorithm, a linear reference path joining the random sample $\boldsymbol{s}_{ref}$ to its corresponding nearest state in the tree $\boldsymbol{s}_{near}$ is used as an input for the lateral dynamic intervention. On the other hand, a random sample $\boldsymbol{s}_{ref}$ is generated and used as an input to the steering controller throughout one iteration in the ARRT algorithm. The four wheel angles $\delta_{fl}(t), \delta_{fr}(t), \delta_{rl}(t), \delta_{rr}(t)$ are continuously calculated, while extending the tree towards this random sample, in closed-loop from the nearest state in the tree $\boldsymbol{s}_{near}$. This extension is performed with the two-track vehicle dynamic model that ensures that the planned trajectory is drivable. If a collision is detected during the extension, the iteration is discontinued and no state is added to the tree. An exception to this is explained in Section 3.5.6 in which a state is added to the tree if a collision is with predicted low severity of injury.

Speed Controller

The steering controller only defines the lateral dynamic intervention, i. e., it just defines how the vehicle should steer while moving towards the sample $\boldsymbol{s}_{ref}$ using a vehicle dynamic model, but the speed controller defines the speed with which it should move. Unlike the CL-RRT algorithm, which uses a single fixed piecewise acceleration profile, the ARRT

algorithm uses multiple piecewise acceleration profiles with the combination of braking, accelerating and constant velocity. This acceleration profiles define the remaining inputs in $\boldsymbol{u}(t)$, i, e., the longitudinal slip values of four tires $s_{l,fl}(t), s_{l,fr}(t), s_{l,rl}(t), s_{l,rr}(t)$.

An example of the acceleration profile $\boldsymbol{a}_x^i$ is shown in Fig. 3.6. The acceleration profile contains three intervals. The first and third interval is either constant braking, constant acceleration or zero acceleration, i. e., constant velocity while the second interval is the linear transition interval between the first and third interval. The duration and the slope of the transition interval are dependent on the difference between the acceleration values in the first and third intervals. The constraint of the maximum allowable jerk j_{max} is considered for defining this transition interval, as shown in the Fig. 3.6.

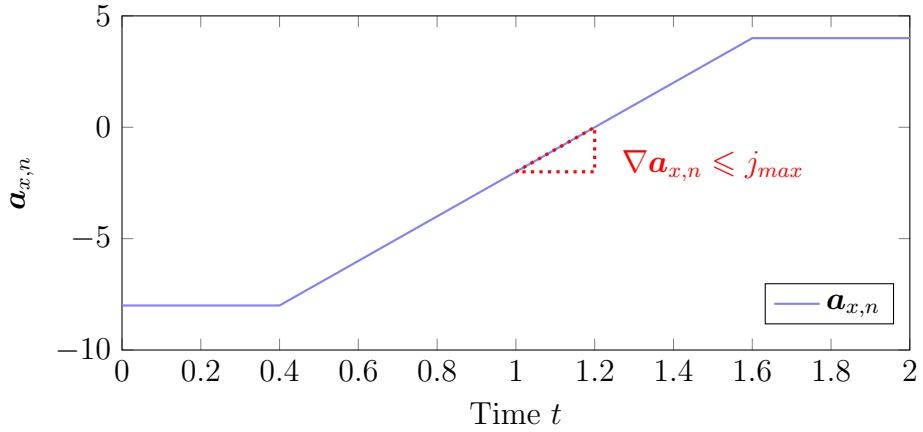


Figure 3.6: Example of an Acceleration Profile.

3.5.4 Goal Selection

In general, the safe trajectory planning task for robotic systems is decomposed into two subtasks. The first can be named as the "find-goal" problem, which aims at defining a safe goal-location for the vehicle [Var93]. The second can be named as "path-planning to a goal" problem, which has the aim to find a path from the current location to the goal location. Algorithms for treating the first task of "find-goal" are mainly based on expert knowledge and are application-specific.

For the problem considered in this work, the goal is actually a region $\boldsymbol{s}_{goal} \subset \boldsymbol{s}_{road}$. The reason behind choosing a region and not a specific goal location is the choice of the ARRT algorithm as a trajectory planning algorithm. It is difficult to reach the exact goal point with the random sampling-based algorithm, such as the ARRT algorithm, especially in a high dimensional continuous state-space. Therefore, the goal region is defined as a circular region with a diameter of 3 m. Also, a fixed goal location is not suitable in a dynamic environment where the time at which the vehicle reaches at the goal is also critical. Therefore, the concept of dynamically changing goals is also introduced as described follow.

The trajectory planning with the ARRT algorithm is divided into two subintervals of a long time interval $[t_0, t_0+\tau]$. An initial goal region S_{goal} is defined for the ARRT algorithm at time t_0 while extending the tree in the first subinterval $[t_0, t_0+\tau_1]$. The center of goal region is chosen from the set of locations, which are the center of lanes, where the center of gravity of the EGO vehicle will lie at the time instance $t_0+\tau$ assuming if it drives alone on the road with constant velocity. The final goal location is chosen based on two factors. First one is the closeness of the lane, on which the goal location lies, to the current lane of the vehicle. T he second factor is if this location is predicted to be collision-free, i. e., whether it lies in $\boldsymbol{s}_{free}(t_0+\tau_1)$. The nearest goal location that is also collision-free is preferred. Otherwise, if no goal location is predicted as collision-free, then the nearest location is chosen. This does not affect the planning of safe trajectories adversely as this is a temporary goal location for the first subinterval. At the start of the second subinterval, the goal location is revised again in a same way based on the final vehicle state found in the first subinterval in order to adapt to the chnages in the dynamic environment. Also, it is not necessary to reach the exact goal location and the aim is to reach in goal region.

3.5.5 Sampling Strategy

The probabilistic completeness of the RRT algorithm is due to the random sampling strategy. However, complete random sampling usually leads to high uncertainty about the time required to converge. Therefore, some bias or deterministic sampling is often introduced to expedite the convergence of the RRT algorithm. The ARRT algorithm also uses a mix of random and deterministic sampling. The use of machine learning algorithms to bias the sampling of the ARRT algorithm and reduce the convergence time further is explained in Chapter 5.

In the CL-RRT algorithm, the sampled input to the controller $\mathbf{r}$ is a combination of a linear path and a fixed speed profile as described in Section 3.4. This limits the capability of the algorithm for finding a safe trajectory in critical traffic-scenarios as the trajectories those can be generated are limited. As explained in Section 3.5.3, the ARRT algorithm uses a sample $\boldsymbol{s}_{ref}$ for the steering controller and it defines multiple acceleration profiles for the speed controller. The sample s $\boldsymbol{s}_{ref}$ is randomly chosen in all iterations. Therefore, it does not require indexing with time. However, it defines the state $\boldsymbol{s}_{near}$ in the tree that is extended in that particular iteration. The input to the speed controller depends on the time of this state and the chosen acceleration profile. This acceleration profile is converted into consecutive segments of time duration Δt and the corresponding segment whose starting time is the same as the time of $\boldsymbol{s}_{near}$, referred to as $\boldsymbol{a}_{ref}(t)$, is used as input for the speed controller. It is important to note that the random sampling of $\boldsymbol{s}_{ref}$ is done only in the region of the road $\boldsymbol{s}_{road}$ between the start state $\boldsymbol{s}(t_0)$ and goal state S_{goal}.

In order to add some deterministic bias to the sampling of $\boldsymbol{s}_{ref}$, the center of the goal state S_{goal} is deterministically sampled at fixed iterations of the ARRT algorithm. This is called *goal-biased sampling*. It helps to keep the direction of the tree extension towards S_{goal}. Also, the possibility of algorithm convergence with goal-biased sampling becomes

higher once it reaches near to S_{goal}. However, the goal-biased sampling is avoided for the first 20 samples (given that the total number of samples is 100) in order to let the ARRT algorithm explore the state space without any bias and then every third sample is used for goal-biased sampling.

3.5.6 Prediction of the Severity of Injury

The prediction of the severity of injuries based on information from exteroceptive sensors before a collision occurs is a new research area. While the potential of exteroceptive sensors for the estimation of the severity of injury is explained in many works, mostly summarized in [KG15], a simple method for the estimation of severity of injury that can be included in a trajectory planning algorithm is missing. Here, an extension is made to the ARRT algorithm to find a trajectory with low severity of injury when a collision is unavoidable or a collision-free trajectory is not found in a specified amount of time.

As explained in Section 3.5.3, when a collision is detected during the extension of the tree, the corresponding robot state is discarded. Instead of rejecting this state, the ARRT algorithm checks the predicted severity and stores the robot state in the tree if its predicted severity of injury is low. Vehicle's delta-v [Jok93], which is a change in velocity between pre-collision and post-collision trajectories of a vehicle, is suitable as the best single predictor of crash severity. However, it requires a detailed model of the crash, which is not the focus of this work. Therefore, a model from [CJ16] is used for estimating the severity of the injury. It has developed the generalized relationship between impact speeds, impact angles, and the severity of injury probability for common crash types based on several simple assumptions. It defines critical impact speeds v_c for different crash types for which the probability of fatal or severe injury is less than 10% rounded to nearest 5 km/h. These critical impact speeds are also shown in Table 3.1. As the nonlinear two-track model is used for estimating vehicle states, the estimated impact velocity and the estimated impact angle are available for every state in the RRT. If the impact speed is less than the critical impact speed, then the trajectory is considered as nonsevere. Nonsevere trajectories are further compared based on the impact velocity, i. e., the trajectories with lower impact speed are given preference. Although the robot states with a predicted nonsevere collision are added to the tree, they are not considered in further iterations for the extension of the tree.

Table 3.1: Approximate Critical Impact Speeds for Common Crash Types.

Crash-type	**Critical impact speed (km/h)**
Pedestrian-vehicle	20
Frontal crash	30
Side crash	30
Rear crash	55

This method used for predicting the severity of injury in this work is based on several simplifications [CJ16]. This is acceptable as the aim of the work is not to provide an accurate crash prediction model. The goal is to provide a possibility to include the desired model in the framework of the ARRT algorithm. For example, a sophisticated model for predicting the severity of injury, such as presented in [MLB+18], can replace the model proposed by [CJ16].

3.5.7 Criteria for Addition of $\boldsymbol{s}_{new}(t + \Delta t)$ to $\mathcal{T}$

The definition of the function f_{obj} is extended from the definition of the indicator function I_{obj} from the Eq. 3.8, such that

$$f_{obj}(t) = \begin{cases} 1, & \text{if } \boldsymbol{s}(t) \bigcap S_{obj}(t) = \varnothing \\ 2, & \text{if } \boldsymbol{s}(t) \bigcap S_{obj}(t) \neq \varnothing \ \wedge \ \mathrm{v}_{r,\boldsymbol{s}(t)} < \mathrm{v}_c \\ 3, & \text{otherwise,} \end{cases} \tag{3.12}$$

where $S_{obj}(t) \subset \mathbb{R}^2$ is the area occupied by N objects, i. e., $\bigcup_1^N \boldsymbol{s}_{obj,n}(t)$ and $\mathrm{v}_{r,\boldsymbol{s}(t)}$ is the predicted relative collision velocity at state $\boldsymbol{s}(t)$ between the EGO vehicle and the collision object. If the collision object is a stationary object then the relative collision velocity $\mathrm{v}_{r,\boldsymbol{s}(t)}$ becomes the absolute velocity of the EGO vehicle. Similarly, the indicator function f_{road} is obtained by expanding the definition of indicator function from Eq. 3.9 at time t is

$$f_{road}(t) = \begin{cases} 1, & \text{if } \boldsymbol{s}(t) \bigcap S_{nr}(t) = \varnothing \\ 2, & \text{if } \boldsymbol{s}(t) \bigcap S_{nr}(t) \neq \varnothing \wedge \ \mathrm{v}_{r,\boldsymbol{s}(t)} < \mathrm{v}_c \\ 3, & \text{otherwise,} \end{cases} \tag{3.13}$$

where, $S_{nr} \subset \mathbb{R}^2$ is the area not belonging to the road. The state $\boldsymbol{s}_{new}$ found in an iteration is added to the tree, if both functions f_{obj} and f_{road} have value of 1 or 2 at each time step within the extension time interval.

3.5.8 Safe Trajectory Selection

The trajectory planning with the ARRT algorithm is divided into two subintervals of a time interval $[t_0, t_0 + \tau]$. In the first subinterval $[t_0, t_0 + \tau_1]$, multiple collision-free trajectories are found by constructing a tree. While choosing the best trajectory, it is important to choose a trajectory that leads to a final vehicle state from which the vehicle can easily be controlled to drive further towards the goal. A maximum steering angle input required for the vehicle to follow the road safely with constant velocity towards the goal-region S_{goal} in the second subinterval $[t_0 + \tau_1, t_0 + \tau]$, from the end position of the trajectories found in the first subinterval, is predicted. This parameter is named *steering effort* and is taken as one of the parameters while choosing the final trajectory.

The ARRT algorithm uses different parameters in preference order safety, steering effort, and acceleration values for the selection of the final trajectory. If it cannot find a collision-free trajectory, then the trajectories with a collision are compared solely based on the

predicted severity of injury and it selects the trajectory with the lowest severity of injury as the best trajectory for that scenario. If multiple collision-free trajectories are found, then only those are considered for further comparison. Among these trajectories, it eliminates the trajectories with high steering effort above the defined threshold or if there is none below the defined threshold, then it selects the one with lowest steering effort as a label. Finally, if multiple collision-free trajectories are remaining, then the one with the lowest maximum absolute acceleration value along each trajectory is selected as the best trajectory.

If it does not find any trajectory without a collision or with nonsevere collision, then the vehicle breaks sharply as a fail-safe.

3.6 Augmented CL-RRT+ Algorithm

The ARRT algorithm provides a suitable framework for finding safe trajectories in complex, critical traffic-scenarios with multiple static and dynamic objects using simultaneous interventions in the lateral and longitudinal dynamics. The lateral dynamic intervention is based on random sampling with uniform distribution, while the longitudinal dynamic intervention is based on the predefined longitudinal acceleration profiles. Although the deterministic nature of longitudinal acceleration profiles provide the possibility to find the trajectories with actuator and comfort constraints, it also simultaneously limits the capability of the algorithm to find safe trajectories in highly complex traffic-scenarios. It is because the limited number of predefined acceleration profiles will not cover all the possible manoeuvres that a vehicle can perform. Therefore, an extension for sampling the longitudinal acceleration is proposed in the ARRT algorithm and the extended algorithm is named as the Augmented CL-RRT+ (ARRT+) algorithm. As shown in the Fig. 3.7 and Alg. 3.3, all other steps in an iteration of the ARRT algorithm remain the same in the ARRT+ algorithm.

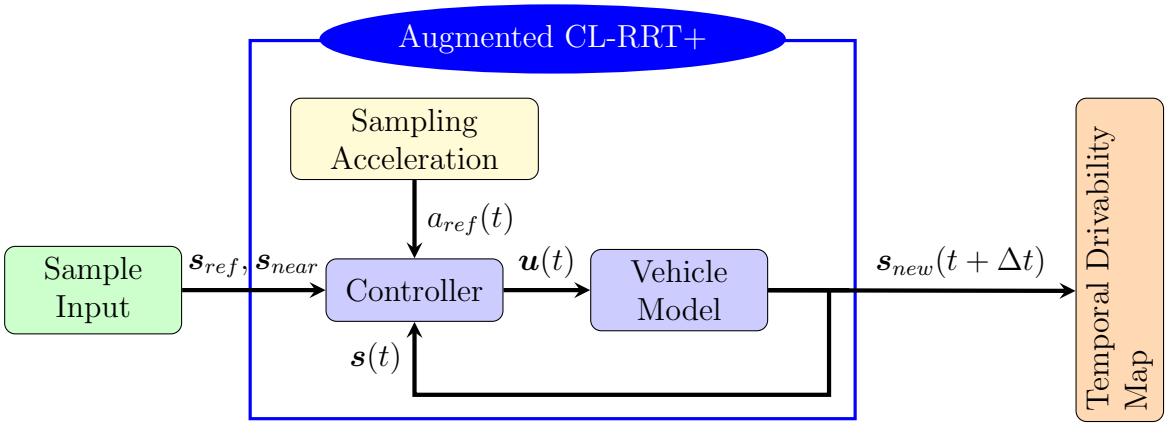


Figure 3.7: Flow Diagram for the Augmented CL-RRT+ Algorithm.

To understand the extension in the ARRT+ algorithm, some additional notations are

Algorithm 3.4 `Augmented CL_RRT+`$(\boldsymbol{s}_{init}, \boldsymbol{v}_{init})$

$S_{goal} \leftarrow$ GOAL_SELECTION $(\boldsymbol{s}_{free}, \boldsymbol{v}_{init})$
for $n = 0$ to N **do**
  $\mathcal{T}_n.init(\boldsymbol{s}_{init})$;
  for $k = 0$ to K **do**
    $\boldsymbol{s}_{rand} \leftarrow$ RANDOM_STATE();
    $\boldsymbol{s}_{ref}, \boldsymbol{s}_{near} \leftarrow$ SAMPLE_INPUT$(\mathcal{T}, \boldsymbol{s}_{rand})$;
    $\boldsymbol{a}_{ref}(t) \leftarrow$ SAMPLE_ACCELERATION$(\boldsymbol{a}_{near})$;
    $\boldsymbol{s}_{new} \leftarrow$ FIND_STATE $(\boldsymbol{s}_{near}, \boldsymbol{s}_{ref}, \boldsymbol{a}_{ref}(t), \Delta t)$;
    $success \leftarrow$ CHECK_CONSTRAINS$(DrivabilityMap)$;
    if $success$ **then**
      $\mathcal{T}_n.add\_vertex(\boldsymbol{s}_{new})$;
      $\mathcal{T}_n.add\_edge(\boldsymbol{s}_{new}, \boldsymbol{s}_{near}, \boldsymbol{u})$;
    else
      $severity \leftarrow$ CHECK_SEVERITY$(\boldsymbol{s}_{new}, \boldsymbol{s}_{obj},)$
      if $severityislow$ **then**
        $\mathcal{T}_n.add\_vertex(\boldsymbol{s}_{new})$;
        $\mathcal{T}_n.add\_edge(\boldsymbol{s}_{new}, \boldsymbol{s}_{near}, \boldsymbol{u})$;
        $\mathcal{T}_n.add\_severity(severity)$;
      end if
    end if
  end for
end for
$\mathcal{T}^* \leftarrow$ COMPARE_TREES$(\mathcal{T}_1, \ldots, \mathcal{T}_N)$
RETURN $\mathcal{T}^*$;

defined. A tree $\mathcal{T}(L) = \{\mathbf{s}_0(t_0), \mathbf{s}_1(t_1), \ldots, \mathbf{s}_L(t_L)\}$ is generated with the ARRT+ algorithm which has L different states with $\mathbf{s}_0(t_0)$ being the initial state at t_0. A number of states may have been extended from the same state in the tree $\mathcal{T}(L)$. Therefore, the tree $\mathcal{T}(L)$ actually comprises of K different trajectories $\boldsymbol{\pi}^k$, for $k = 1, 2, \ldots, K$, which are either collision-free or with a predicted nonsevere collision. In order to explain the strategy for the sampling of the longitudinal acceleration, all trajectories are represented as $\boldsymbol{\pi}^k = \{\mathbf{r}_0, \mathbf{r}_1^k, \ldots, \mathbf{r}_I^k\}$, where $\mathbf{r}_{i-1}^k$ is the parent state of $\mathbf{r}_i^k$, for $i = 1, 2, \ldots, I$. The first state $\mathbf{r}_0$ in every trajectory $\boldsymbol{\pi}^k$ is the initial state $\mathbf{s}_0(t_0)$ in tree $\mathcal{T}(L)$ and $\mathbf{r}_i^k \in \mathcal{T}(L)$, for $i = 0, 1, \ldots$. An example of such representation is shown in Fig. 3.8.

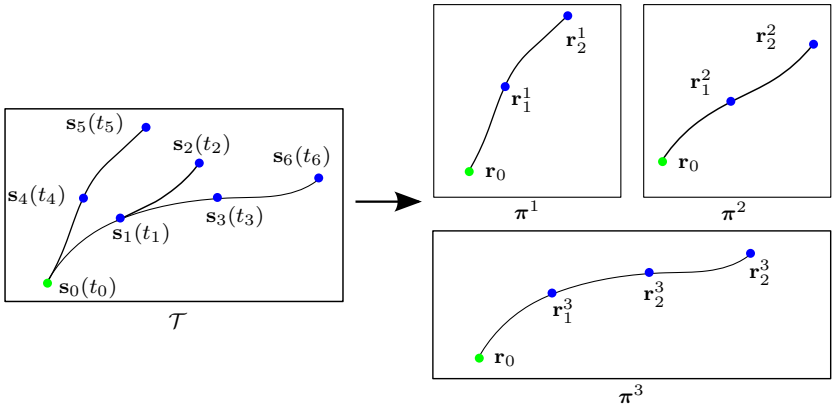


Figure 3.8: Tree $\mathcal{T}(L)$ to Trajectories $\boldsymbol{\pi}^k$.

As every state $\mathbf{r}_i^k$ is obtained by extension from its parent state $\mathbf{r}_{i-1}^k$ for time Δt, the time parameter $t_{\mathbf{r}_i^k}$ of the state $\mathbf{r}_i^k$ is

$$t_{\mathbf{r}_i^k} = t_{\mathbf{r}_{i-1}^k} + \Delta t. \tag{3.14}$$

3.6.1 Sampling Strategy for the ARRT+ Algorithm

The ARRT+ algorithm samples two random variables to find a state $\mathbf{r}_i^k$. First, it samples the type of accelerations $\lambda_{sample} \in \{\lambda^-, \lambda^+, \lambda^c, \lambda^0\}$, where $\lambda^-, \lambda^+, \lambda^c, \lambda^0$ are negative, positive, constant, and zero acceleration, respectively. Here, λ^- has the highest sampling probability followed by λ^c, λ^0, and λ^+. The algorithm assigns this sampled acceleration λ_{sample} to the type of the acceleration $\lambda_{\mathbf{r}_i^k}$ of the state $\mathbf{r}_i^k$, if $\lambda_{\mathbf{r}_{i-2}^k} = \lambda_{\mathbf{r}_{i-1}^k}$ or $\lambda_{\mathbf{r}_{i-1}^k} = \lambda_{sample}$ or $\lambda_{\mathbf{r}_{i-1}^k} = \lambda^0$. This condition It repeats this procedure till any one of this condition is satisfied. This constraint is called as the *stable profile constraint*. This ensures that the EGO vehicle will not accelerate or decelerate alternatively in small time intervals and same acceleration type is used for at least two consecutive states. For states $\mathbf{r}_i^k$ whose parent node is $\mathbf{r}_0$, it does not consider the stable profile constraint as only one past state is present.

Followed by finding $\lambda_{\mathbf{r}_i^k}$, the ARRT+ algorithm samples the acceleration value $a_{\mathbf{r}_i^k}$ of the state $\mathbf{r}_i^k$ uniformly within $a_{\lambda_{\mathbf{r}_i^k},min}$ and $a_{\lambda_{\mathbf{r}_i^k},max}$, where $a_{\lambda_{\mathbf{r}_i^k},min}$ and $a_{\lambda_{\mathbf{r}_i^k},max}$ are the minimum and the maximum acceleration limits for the $\lambda_{\mathbf{r}_i^k}$, respectively. This ensures that the acceleration is sampled within actuator limits for the selected $\lambda_{\mathbf{r}_i^k}$. The algorithm further calculates the acceleration values $\boldsymbol{a}_{\mathbf{r}_{i-1}^k,\mathbf{r}_i^k}$ for travelling from the state $\mathbf{r}_{i-1}^k$, i. e., for a time interval $[t_{\mathbf{r}_{i-1}^k}, t_{\mathbf{r}_i^k}]$ using an exponential interpolation function of time to get smooth transition behaviour between states. It derives the resulting jerk values from $\boldsymbol{a}_{\mathbf{r}_{i-1}^k,\mathbf{r}_i^k}$ and checks if they are less than the maximum allowed jerk. Actuator constraints consist of these acceleration and jerk constraints. If any of the stable profile or the actuator constraints is not satisfied, then the procedure is repeated from the start. On the other hand, if all the constraints are satisfied, $\boldsymbol{a}_{\mathbf{r}_{i-1}^k,\mathbf{r}_i^k}$ along with s_{rand} is used as the input to find the state $\mathbf{r}_i^k$.

3.7 Comparison Between the ARRT and the ARRT+ Algorithm

Many critical traffic-scenarios on curved road are simulated in the Matlab simulation environment explained in Section 2.4, by changing initial velocities and initial positions of road traffic-participants. These scenarios are broadly divided into two categories, 4-object and 6-object scenarios, based on the number of objects, including the EGO vehicle, in the traffic-scenario. For both algorithms, 2100 samples ($\boldsymbol{s}_{rand}$) are used to find safe trajectories in all of these scenarios. The results of the simulation are shown in Table 3.2.

Table 3.2: Comparison Between the ARRT and the ARRT+ Algorithm.

	4-object Scenario (405 Scenarios)		**6-object Scenario** (478 Scenarios)	
Criteria	**ARRT**	**ARRT+**	**ARRT**	**ARRT+**
Average # States	595	210	629	201
Average Time (Sec.)	6.11	3.59	6.76	4.19
Collision-free Trajectory Found (%)	98.76	96.79	77.84	90.37
No Safe Trajectory Found (%)	0	0	0.05	0.09

The results show that the ARRT+ algorithm needs less memory (number of robot states to be stored) and less time than the ARRT algorithm for finding a safe trajectory. Thus, it is more efficient in terms of both memory and time. In terms of finding a safe trajectory, it is significantly better than the ARRT algorithm in 6-object scenarios while slightly worse in 4-object scenarios. It is due to the increase in the state-space for the random sampling in the ARRT+ algorithm, which makes it challenging to find collision-free trajectories in some of the complex traffic-scenarios with the limited number of samples.

Summary

This chapter presented a literature survey of the trajectory planning algorithms along with their limitations, especially for the problem of safe trajectory planning in critical traffic-scenarios. Further, it formulated a problem definition for the trajectory planning task in critical traffic-scenarios and defined the procedure for the detection of the critical traffic-scenarios. It also presented two variants of trajectory planning algorithms, namely Augmented CL-RRT and Augmented CL-RRT+, along with details of their extensions to the original CL-RRT algorithm. Finally, the comparison results based on the simulation of critical traffic-scenarios with these algorithms show that the Augmented CL-RRT+ algorithm is more efficient and works better in most of the traffic- scenarios.

Chapter 4

Machine Learning

The field of machine learning is about computers learning a class of tasks without explicitly programming their solution but improving their accuracy by feeding the data in the form of experiences [T.06]. Owing to the advancements in this area, it's applications are growing exponentially in different disciplines such as agriculture [LBM+18], finance [dP18], manufacturing [WWIT16], etc. However, they are still not preferred in safety-critical applications such as vehicle trajectory planning. Therefore, the focus of this work is the development of hybrid machine learning algorithms, which are a combination of machine learning algorithms with physical models, so that they can be used in safety-critical applications as well. This chapter provides a brief description of the basics of machine learning algorithms necessary to understand the proposed hybrid machine learning algorithms in Chapter 5.

The outline of the chapter is as follows: Section 4.1 describes the basic concepts in machine learning. Section 4.2 explains the different types of supervised neural network algorithms used in this work. Further, Section 4.3 illustrates unsupervised machine learning algorithms such as clustering algorithms, autoencoder, variational autoencoder.

4.1 Basic Concepts in Machine Learning

4.1.1 Types of Machine Learning

Machine learning algorithms are broadly classified into three categories. The area of machine learning in which the output y_i is known for the corresponding input $\boldsymbol{x}_i \in \mathbb{R}^n$, also called as *feature space*, in the form of input-output pairs $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ to guide the learning is called *supervised learning*. Here, $\mathcal{D}$ is called the *training set* and N is the total number of input-output pair samples. The supervised learning is further divided into two types based on the type of output variable. If y_i is a categorical or nominal variable from some finite set such that $y_i \in \{c_1, c_2, \ldots, c_K\}$, where K is the total number of *classes* or

labels, then it is a task of *classification* to produce a function $f : \mathbb{R}^n \to \{c_1, c_2, \ldots, c_K\}$. On the other hand, when $y_i \in \mathbb{R}$, i. e., it is a real-valued scalar, then the task is termed as *regression*. In such cases, the task of a machine learning algorithm is to create a function such that $f : \mathbb{R}^n \to \mathbb{R}$.

The second type of machine learning is *unsupervised learning*. Here only inputs, i. e., $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$ are given while the labels are unknown. The goal is to find inherent patterns in the data. Clustering similar data, anomaly detection, density estimation, generation of new data samples are some of the applications of unsupervised learning.

The third type is *reinforcement learning*, which deals with learning agents in an environment, such that an agent gains a maximum cumulative reward over a fixed time interval or a number of steps. It differs from supervised learning as the input-output pairs are not given, but the agent finds them through exploration. Also, reinforcement learning is not concerned with finding patterns in the data like unsupervised learning.

Primarily, the proposed hybrid machine learning algorithms use supervised and unsupervised learning algorithms. The comparison of this proposed methodology to the framework of reinforcement learning is described in Section 5.4.6 and the possibility of extending those methodologies with reinforcement learning algorithms is explained in Chapter 7.

4.1.2 Prediction Error

The mapping function f between the input feature vector $\boldsymbol{x}$ and it's corresponding label y is often not deterministic [Vid97]. Therefore, a statistical framework to represent the stochasticity is defined with x and y as the random variables for $\boldsymbol{x}$ and y, respectively. The joint probability distribution $p(\mathsf{x} = \boldsymbol{x}_i, \mathsf{y} = c_k)$ provides the complete information about the uncertainty associated with random variables x and y. In order to evaluate the different mapping functions f and choose the suitable among them, a performance measure has to be defined. Only reducing the misclassification rate is not a suitable method for performance measurement as the consequences for two different misclassifications can be different. Therefore, a *loss function* $\mathcal{L}(y, \hat{y})$, which assigns a cost to the prediction $\hat{y} = f(\boldsymbol{x})$ given the true prediction y, is initially defined. In regression task it is usually either the absolute error $\mid \mathsf{y} - f(\mathsf{x}) \mid$ or the squared error $\mid\mid \mathsf{y} - f(\mathsf{x}) \mid\mid^2$ whereas in classification task it is commonly defined as 0/1-loss

$$\mathcal{L}(\mathsf{y}, f(\mathsf{x})) = \begin{cases} 0, & \text{if } f(\mathsf{x}) = \mathsf{y} \\ 1, & \text{otherwise.} \end{cases} \tag{4.1}$$

The performance is measured in terms of a prediction error which is the expectation of $\mathcal{L}(y, f(\boldsymbol{x}))$ over x, y-space. For regression task the prediction error is calculated as

$$\mathcal{E}(f) = E_{\mathsf{x},\mathsf{y}}\{\mathcal{L}(\mathsf{y}, f(\mathsf{x}))\} = \int_{\mathbb{R}^N} \int_{\mathbb{R}} \mathcal{L}\left(y, f(\boldsymbol{x})\right) p(\mathsf{x} = \boldsymbol{x}, \mathsf{y} = c) dy d\boldsymbol{x}. \tag{4.2}$$

For the classification task, this is defined as

$$\mathcal{E}(f) = E_{\mathbf{x},\mathsf{y}}\{\mathcal{L}(\mathsf{y}, f(\mathbf{x}))\} = \int_{\mathbb{R}^N} \sum_{k=1}^{K} \mathcal{L}\left(y, f(\boldsymbol{x})\right) p(\mathbf{x} = \boldsymbol{x}, \mathsf{y} = c_k) d\boldsymbol{x}. \tag{4.3}$$

Although the above mentioned loss functions are used only for supervised learning algorithms, they are useful also in reinforcement learning or unsupervised learning. For example, Q-learning, which is a type of reinforcement learning algorithm, predicts the expected scalar reward that can be earned by taking a particular action using supervised learning algorithms. The loss function of regression tasks is used here to evaluate the performance of the algorithm. In autoencoders, a type of unsupervised learning algorithms, a reconstruction loss equivalent to the regression loss is used. The only difference is that the input is itself output. Therefore, no additional output labels are necessary.

4.1.3 Bayes Classifier

The optimal solution of machine learning for the classification task is the one which minimizes the prediction error $\mathcal{E}(f)$ such that

$$f_B = \underset{f}{\operatorname{argmin}}\{\mathcal{E}(f)\}, \tag{4.4}$$

where the function f_B is called as *Bayes classifier*.

The mapping function f in case of classification task is a rule set which makes partitions of the input space $\mathbb{R}^N$ into regions $\mathcal{R}_k$, such that all points in $\mathcal{R}_k$ belong to class c_k while simultaneously no points belonging to class c_k lie outside of region $\mathcal{R}_k$. Therefore, the prediction risk in Eq. 4.3 can be rewritten as

$$\mathcal{E}(f) = E_{\mathbf{x},\mathsf{y}}\{\mathcal{L}(\mathsf{y}, f(\mathbf{x}))\} = \sum_{k=1}^{K} \sum_{l=1}^{K} \int_{\mathcal{R}^k} \mathcal{L}(c_k, c_l) p(\mathbf{x} = \boldsymbol{x}, \mathsf{y} = c_k) d\boldsymbol{x}. \tag{4.5}$$

Here, $\mathcal{L}(\mathsf{y}, f(\mathbf{x}))$ is the cost for assigning a class c_l to the input vector $\mathbf{x}$ when the true class is c_k. The goal of the classification task is to choose the regions $\mathcal{R}_k$ such that the expected loss defined in Eq. 4.5 is minimized. It implies that for each $\mathbf{x}$, $\sum_{l=1}^{K} \mathcal{L}(c_k, c_l) p(\mathbf{x} = \boldsymbol{x}, \mathsf{y} = c_k)$ should be minimized. This leads to a Bayes classifier

$$f_B(x) = \underset{c_l}{\operatorname{argmin}} \Big\{ \sum_{k=1}^{K} \mathcal{L}(c_k, c_l) p(\mathbf{x} = \boldsymbol{x}, \mathsf{y} = c_k) \Big\}. \tag{4.6}$$

4.1.4 Curse of Dimensionality

The task of generalizing with a finite number of samples for machine learning algorithms becomes more difficult with the increase in the dimension of feature space, with each feature having a range of values. A huge amount of training data is needed to have enough samples

with each combination of values. The sampling density is proportional to $N^{(1/D)}$, where N is the size of the training set and D is the dimension of the feature space. Thus, the feature space becomes sparser with the addition of a dimension to the feature space. This phenomenon is called as *curse of dimensionality*.

4.1.5 Model Selection

The model for classification tasks should be selected, which gives the least prediction error as defined in Eq. 4.3. However, the probability density functions are normally not known in practical applications. Therefore, an empirical prediction error based on the available training data $\mathcal{D}$

$$\mathcal{E}_{emp}(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, f(\boldsymbol{x}_i)), \tag{4.7}$$

is used. However, more important is the *generalization error*, which is the expected prediction error for untrained or previously unseen samples. Therefore, the generalization error is approximated with an independent *test dataset*, which is not used for training the model. It is possible that the model used for training overfits or underfits the training data $\mathcal{D}$, which means the model is not able to obtain sufficiently low generalization error. This generalization error can be controlled by altering the *model capacity*, which describes how complex the mapping is, that it can learn. Machine learning algorithms perform best when the capacity of the model is appropriate to the true complexity of the task they need to perform and the amount of the data provided.

Fig. 4.1 shows the relation between the model capacity and errors. The training and test error behave differently with the increase in the model capacity. As the model capacity increases, both training error and test error decrease to a minimum test error. This is the *underfitting zone*. With a further increase in the model capacity, the training error decreases, but the test error increases. It is because the model has a larger capacity than the optimal capacity and it overfits the noise in training data. Therefore, this zone is known as the *overfitting zone*.

The capacity of the model generally depends on the number of parameters used in the model. These parameters are divided into two types: *hyperparameters* and the *learned parameters* from data. The hyperparameters are specific settings to control the behaviour of the model that are not learned on training data. Otherwise, it will always choose a model capacity that will result in overfitting. To find the optimal set of hyperparameters, a *validation set* of samples not included in the training set and test set is used.

4.1.6 Bias and Variance

To better understand its different sources and how it can be reduced, the prediction error $\mathcal{E}(f)$ is decomposed into three types of errors, namely bias $\mathcal{E}_b(f)$, variance $\mathcal{E}_v(f)$ and a quantity called irreducible error resulting from the noise. Given a particular learning

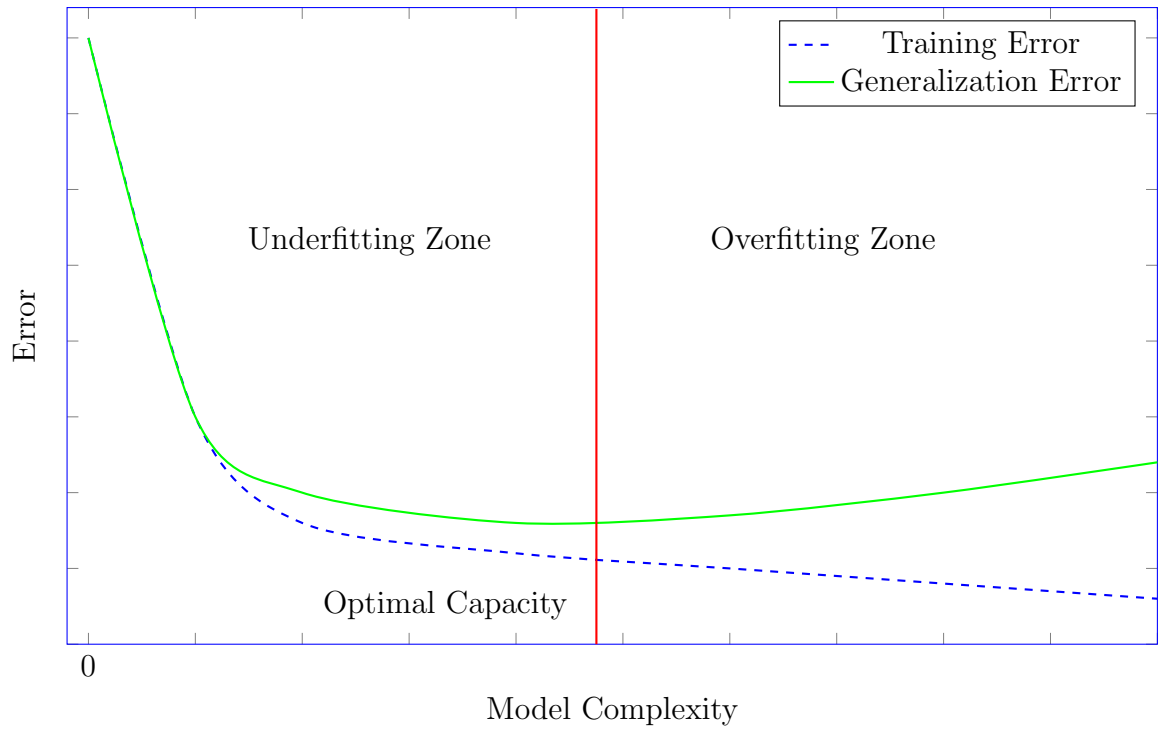


Figure 4.1: Relationship Between Model Capacity and Error.

algorithm, only bias and variance can be influenced by altering the design of an algorithm to minimize the prediction error. However, these two terms are not independent and a lower bias results in higher variance and vice versa. This conflict of (supervised) learning algorithms trying to minimize both these errors simultaneously to generalize well beyond the training set is termed as the *bias-variance tradeoff*.

Bias results from the erroneous assumption about the capacity of the learning model. The error due to bias is taken as the difference between the expected prediction of learning model $f_{\mathsf{D}}(\mathbf{x})$[1] and the bayes classifier $f_B(\mathbf{x})$ output considered as the correct value. In other words, the learning model has high bias if the prediction of the model $f_{\mathsf{D}}(\mathbf{x})$ differs significantly on an average over the datasets, generated from same distribution, from the predictions of the optimal model $f_B(\mathbf{x})$. Mathematically, the bias error $\mathcal{E}_b(f)$ is expressed as

$$\mathcal{E}_b(f) = \mathcal{L}(f_B(\mathbf{x}), f_{\mathsf{D}}(\mathbf{x})). \tag{4.8}$$

The estimator is said to be unbiased if the bias error $\mathcal{E}_b(f)$ is zero which implies $f_B(\mathbf{x}) = E(f_{\mathsf{D}}(\mathbf{x}))$. Similarly, another property of the model important to consider is its variability for a given data sample. This is the error due to variance $\mathcal{E}_v(f)$. A model has high variance if the prediction of the model $f_{\mathsf{D}}(\mathbf{x})$ trained on different datasets, generated from the same

[1]Here, the learning model is defined as $f_{\mathsf{D}}(\mathbf{x})$ instead of $f(\mathbf{x})$ as the suffix D indicates random training dataset D)

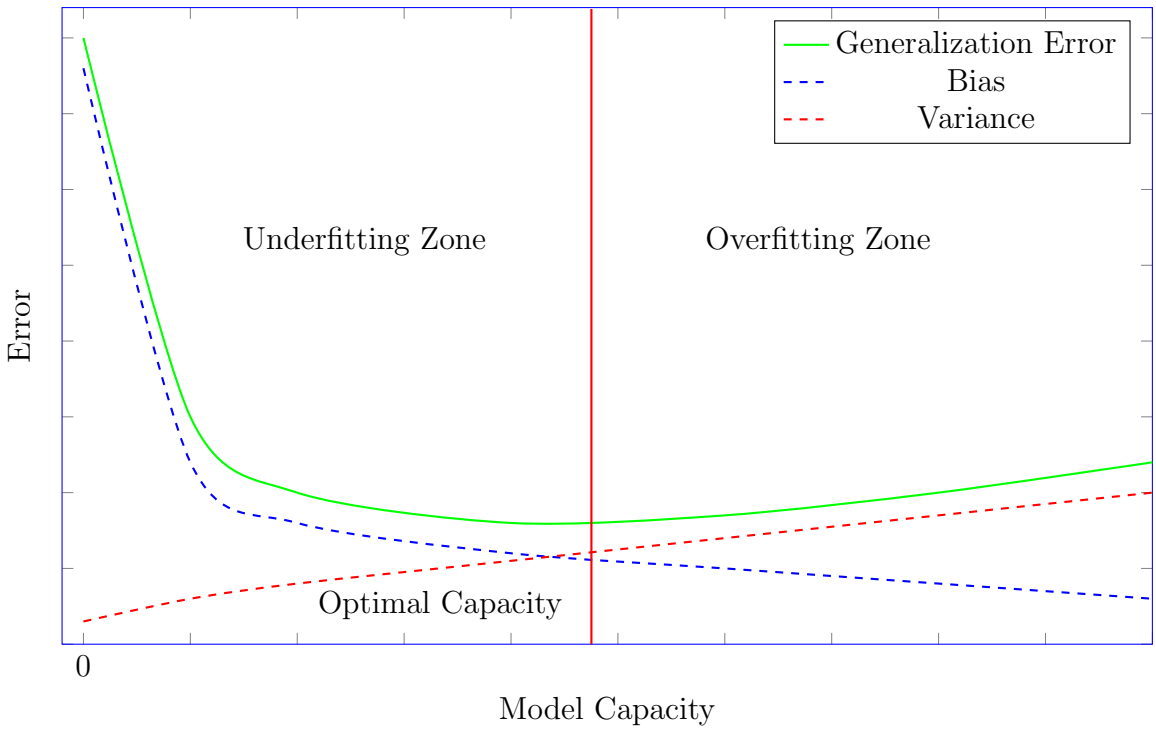


Figure 4.2: Relationship Between Model Capacity, Bias and Variance.

distribution, differs significantly from its expectation. This is expressed in equation as

$$\mathcal{E}_v(f) = E_{\mathsf{D}}\left(\mathcal{L}(f_B(\mathbf{x}), f_{\mathsf{D}}(\mathbf{x}))\right). \tag{4.9}$$

Intuitively, high bias indicates underfitting and high variance indicates overfitting. Therefore, the desirable models are those which manage to keep both bias and variance low. Fig. 4.2 shows the relation of the model capacity with bias and variance.

The bias-variance decomposition for classification is different from that of regression as the loss function is defined differently. The different possible bias-variance decompositions are explained in [Dom00].

4.1.7 No Free Lunch Theorem

The machine learning algorithms are designed to find the best approximation of the mapping function f such that the generalization error is small. The *No Free Lunch Theorem* states that every classification algorithm has the same average error rate over all possible data generating distributions, which means there is no single model that performs best for every problem [WM97]. A model is superior compared to others in a specific domain only because the assumptions fit particularly well to the data generating distribution of that domain.

As a consequence of the no free lunch theorem, the aim of machine learning is not to develop a universal learning algorithm, but to develop different types of models for different data distributions that occur in the real world. Also, for each model, there are different algorithms for training which make different speed, accuracy, complexity trade-offs. Therefore, different models have to be evaluated to find their suitability to the problem at hand.

4.1.8 Evaluation of Machine Learning Algorithms

Confusion Matrix

The confusion matrix is a two-dimensional table used for the visualization of classifier performance. Each row and column of the matrix represents the total instances in an actual class and predicted class, respectively. The name stems from the fact that it helps to understand quickly if the classifier is confusing between classes. Table 4.1 shows a confusion matrix for two classes c_1 and c_2.

	$\hat{y} = c_1$	$\hat{y} = c_2$
$y = c_1$	TN	FP
$y = c_2$	FN	TP

Table 4.1: Confusion Matrix for Classes c_1 and c_2.

If c_1 is considered as a negative class nad c_2 is considered as a positive class, then the four elements in clockwise direction from upper left corner of the table are defined as true negative (TN), false positive (FP), true positive (TP), and false negative (FN). The metrics *true positive rate* (TPR) or *sensitivity* and *false positive rate* (FPR) can be evaluated as

$$TPR = \frac{TP}{FN + TP}. \tag{4.10}$$

$$FPR = \frac{FP}{TN + FP}. \tag{4.11}$$

Also, the estimated accuracy AC of the classifier, which treats both false positive and false negative error equally, can be found out as

$$AC = \frac{(TN + TP)}{(FN + FP + TN + TP)}. \tag{4.12}$$

If a false positive or false negative error does not cost equally, such that $FN = \alpha_{ROC} FP$, where $\alpha_{ROC} \neq 1$, then a threshold has to be found to adjust the number of false positives and false negatives such that the total cost is low. Therefore, with different α_{ROC} different threshold have to be found.

4.1.9 Receiver Operating Characteristic

The ROC curve was first developed by electrical engineers and radar engineers during World War II for the detection of enemy objects in battlefields. Since then, ROC analysis has been used in many other fields, including machine learning research. The ROC-curve provides a way to study the FN-FP trade-off without having to choose a specific threshold. It is a curve obtained by plotting TPR against FPR using a different set of thresholds. An example of the ROC curve is shown in Fig. 4.3. Any system can achieve the point on the bottom left, $(FPR = 0, TPR = 0)$, by setting $\alpha_{ROC} = 1$ and thus classifying everything as negative; similarly any system can achieve the point on the top right, $(FPR = 1, TPR = 1)$, by setting $\alpha_{ROC} = 0$ and thus classifying everything as positive. If a system is performing randomly, then any point on the diagonal line $TPR = FPR$ by choosing an appropriate threshold can be achieved. A system that perfectly separates the positives from negatives has a threshold that can achieve the top left corner, $(FPR = 0, TPR = 1)$; by varying the threshold, such a system will coincide with the left axis and then the top axis.

Cross-Validation

The division of the dataset into a fixed training set and a fixed test set can be problematic if it results in the test set being small. A small test set implies statistical uncertainty around the estimated average test error, making it difficult to claim which algorithm works better on the given task.

When the dataset has thousands of examples, this is not a severe issue. When the dataset is too small, there are alternative procedures, which allow one to use all of the examples in the estimation of the mean test error, at the price of increased computational cost. These

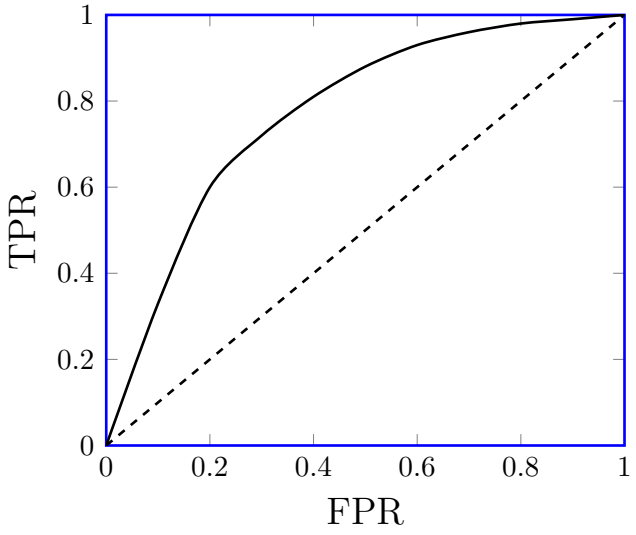


Figure 4.3: Receiver Operating Characteristic Curve.

procedures are based on the idea of repeating the training and testing computation on different randomly chosen subsets or splits of the original dataset. The most common of these is the k-fold cross-validation procedure, in which a partition of the dataset is formed by splitting it into k non-overlapping subsets. The test error may then be estimated by taking the average test error across k trials. On trial i, the i^{th} subset of the data is used as the test set and the rest of the data is used as the training set.

top-i Classification Error

A machine learning algorithm, namely neural networks, predicts a probability distribution over all the target variables. In this case, the target variable with highest probability is considered as predicted output. The *top-i classification error* a criterion to measure the classification error that uses this property. It tests if the reference class is within i hypotheses with the highest probability. This criterion is suitable when multiple labels are to be predicted.

4.1.10 Maximum Likelihood Estimation

Before the evaluation of machine learning algorithms, it is necessary to determine the values of parameters used in the model. The most common principle used for this is the *Maximum Likelihood Estimation* (MLE). It is a method for finding the parameter $\boldsymbol{\theta}$ values such that they maximize the likelihood of producing data $\boldsymbol{x}$ by the model from the observed data distribution $f_{data}(\mathbf{x})$. The MLE for $\boldsymbol{\theta}$ of observing all data points, i. e., the joint probability distribution of all observed data points

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p_{model}(\mathbf{x}|\boldsymbol{\theta}), \tag{4.13}$$

where $p_{model}(\boldsymbol{x}|\boldsymbol{\theta})$ is the model distribution given parameters $\boldsymbol{\theta}$. However, generally finding a joint probability distribution is difficult. Therefore, it is assumed that each generated data is independent of each other and the total probability of observing all data is the product of observing each data point individually. Thus, the maximum likelihood can be modified as

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p_{model}(\boldsymbol{x}_i|\boldsymbol{\theta}), \tag{4.14}$$

where N is the total number of data samples. Still, the product over many probabilities is inconvenient. To obtain an equivalent but simpler optimization problem, the logarithm of likelihood is taken to convert the product into sum such that

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p_{model}(\boldsymbol{x}_i|\boldsymbol{\theta}). \tag{4.15}$$

Although the objective functions are different, the parameters which maximize both objective functions are the same. By rescaling the objective function with $\frac{1}{N}$, the above

expression can be can be interpreted as an approximation of the expectation with respect to the data distribution $p_{data}(\mathbf{x})$ defined as

$$\boldsymbol{\theta}_{ML} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, E_{\mathbf{x}\sim p_{data}(\mathbf{x})} p_{model}(\mathbf{x}|\boldsymbol{\theta}). \tag{4.16}$$

From this equation, it can be interpreted that the MLE tries to minimize the dissimilarity between the observed data distribution $p_{data}(\mathbf{x})$ and the model distribution $p_{model}(\mathbf{x};\boldsymbol{\theta})$. The degree of the dissimilarity between these two is measured by the Kullback-Leiber (KL) Divergence D_{KL}

$$D_{KL}(p_{data}(\mathbf{x}) \parallel p_{model}(\mathbf{x})) = E_{\mathbf{x}\sim p_{data}(\mathbf{x})}(\log p_{data}(\mathbf{x}) - \log p_{model}(\boldsymbol{x}_i)). \tag{4.17}$$

This indicates in order to minimize the KL divergence, i. e., to minimize the difference between the $p_{data}(\boldsymbol{x})$ and $p_{model}(\boldsymbol{x})$, $-E_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})} p_{model}(\boldsymbol{x})$ has to be minimized which is equivalent to the maximization of $E_{\boldsymbol{x}\sim p_{data}} p_{model}(\boldsymbol{x})$ as is Eq. 4.16. Thus, MLE leads to the minimization of the negative log-likelihood.

4.1.11 Motivation for Deep Learning

The challenge of generalizing with training traditional machine learning algorithms for high dimensional data such as image, speech, text data did not prove successful. One thing to notice in such a kind of data is the presence of a specific input structure. For example, the images have a 2D spatial structure and speech or text have a temporal structure. Deep learning algorithms take advantage of this knowledge about the data and encode these beliefs into the model architecture. It lets them define a strong prior for model parameters, which play a crucial role in determining where the parameters end up.

4.2 Supervised Learning Algorithms

There are many types of supervised machine learning algorithms, such as support vector machines, decision trees, random forest, etc. However, only the basics of neural networks are described in this section as the methodologies proposed in this work focus on only the usage of neural networks.

4.2.1 Feedforward Neural Networks

A neural network (NN) or feedforward neural network or multilayer perceptron consists of many connected processors called neurons each producing real-valued activations. The goal of NN is to approaximate a function $\boldsymbol{f}$ which maps the information from input neurons $\mathbf{x}$ to the probability distribution $\boldsymbol{p}(\hat{\mathbf{y}}|\mathbf{x})$, unlike directly to the class as described in Section 4.1, over output labels for the classification task such that

$$\boldsymbol{p}(\hat{\mathbf{y}}|\mathbf{x}) = [p(\mathrm{y} = c_1|\mathbf{x} = \boldsymbol{x}), \ldots, p(\mathrm{y} = c_K|\mathbf{x} = \boldsymbol{x})]^T. \tag{4.18}$$

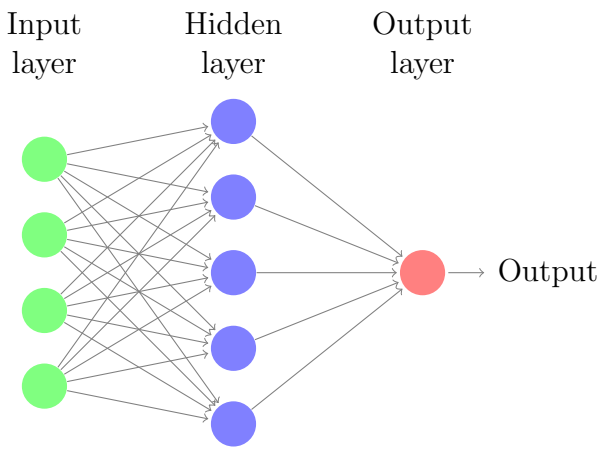


Figure 4.4: Feedforward Neural Network with a Single Hidden Layer.

These models are called *feedforward* as the information flows through the mapping function $\boldsymbol{f}$ from input $\mathbf{x}$ to the output $p(\hat{\mathbf{y}}|\mathbf{x})$ activating other neurons in the process. There are no feedback connections in which the output of the model is fed back into itself.

The mapping function $\boldsymbol{f}$ in the NN basically composed of many different functions with a directed acyclic graph in the form of a chain. Therefore, they are called *networks*. For example the NN function $\boldsymbol{f}(\mathbf{x}) = \boldsymbol{f}^3(\boldsymbol{f}^2(\boldsymbol{f}^1(\mathbf{x})))$ is a three layer NN with $\boldsymbol{f}^1$ describing the first layer of NN having H neurons. Thus, the transformation of input $\mathbf{x} \in \mathbb{R}^N$ at first layer into a vector $\boldsymbol{f}^1 \in \mathbb{R}^H$

$$\boldsymbol{f}^1 = \boldsymbol{\sigma}^1(\boldsymbol{W}^1\mathbf{x} + \boldsymbol{b}^1), \tag{4.19}$$

where $\boldsymbol{W}^1 \in \mathbb{R}^{H\times N}$ is the weight matrix with the (h, n^{th}) element of the matrix defining the weighted connection between the h^{th} neuron in the first layer and n^{th} value in input $\mathbf{x}$, $\boldsymbol{b}^1 \in \mathbb{R}^H$ is the bias and $\boldsymbol{\sigma}^1$ is a nonlinear activation function. In general the transformation in each layer of a NN with L layers can be defined as

$$\boldsymbol{f}^l = \boldsymbol{\sigma}^l(\boldsymbol{W}^l\boldsymbol{f}^{l-1} + \boldsymbol{b}^l), \tag{4.20}$$

where $\boldsymbol{W}^l$ is the weighted connection between the l^{th} and $(l-1)^{th}$ layer, $\boldsymbol{b}^l$ is the bias vector in the l^{th} layer while $\boldsymbol{f}^0$ and $\boldsymbol{f}^L$ defines the input $\mathbf{x}$ and output $\boldsymbol{p}(\hat{\mathbf{y}}|\mathbf{x})$, respectively.

The overall number of layers gives the depth of the model. It is from this the terminology *deep learning* arises. The mapping function $\boldsymbol{f}(\mathbf{x})$ only defines that the final layer, i. e., the output layer of a NN must produce the corresponding output $\mathbf{y}$. It does not define the behaviour of intermediate layers. Therefore, these layers are called hidden layers. Fig. 4.4 shows a NN with one hidden layer.

Finally, these networks are called *neural* because they are loosely inspired by neuroscience. Each element of the hidden layer may be interpreted as playing a role analogous to a neuron in the brain in the sense that it receives inputs from many other units and computes its own activation value. However, many mathematical and engineering disciplines guide modern

NN research. Thus, although NNs take inspiration from how the brain works, it is not the model of the brain.

Most of the NNs are defined with three components: a loss function, an optimization procedure and the architecture. Any of these components can be replaced independently. Therefore, a vast number of variants of NNs can be obtained. Finding a suitable combination of these components to achieve high generalization performance is the aim of building a NN. In this section, these components are discussed.

Loss Functions

Similar to other machine learning algorithms, the training of a neurual network aims to find values for the weight parameters $\boldsymbol{W}$ and the bias parameters $\boldsymbol{b}$ such that the loss function $\mathcal{L}(\mathbf{y}, \boldsymbol{f}(\mathbf{x}))$ is reduced. These parameters are learned using the backpropagation procedure described in Appendix 8.2. In most cases, a NN for classification outputs a probability distribution $p(\hat{\mathbf{y}}|\mathbf{x})$ instead of just correct label $\mathbf{y}$. Therefore, the principle of maximum likelihood estimation is used which means the cross-entropy between the training data and model predictions are used as cost function. A one-hot encoded vector $\mathbf{y}$ is defined as the vector with the value one at position k and zeros elsewhere, given the target value is the class c_k. A regularization term such as the weight decay is frequently added to the cost function to avoid overfitting.

Optimization of NNs

Traditionally, optimization algorithms aim at reducing the defined cost function to a low value. The optimization algorithms used for the training of NNs and, in general, machine learning algorithms act indirectly because the goal for machine learning algorithms is to reduce the generalization error. If the true data distribution is known, then the expectation can be taken over it and traditional optimization procedures can be applied. However, the true data distribution is generally not known and only the training set of limited samples is available. Therefore, machine learning algorithms convert the problem into an optimization task of minimizing the expected loss on the training set using the empirical risk minimization as in Eq. 4.7. However, this is prone to overfit as models will simply try to memorize the training set.

The use of a surrogate loss function and early stopping avoids the problem of overfitting. The surrogate loss function is an alternative loss function than the actual loss function, which has more advantages. For example, a negative likelihood of a correct class is used as a surrogate for 0-1 loss, which is comparatively easier to optimize and it continues to make the model robust by pushing classes apart even when the 0-1 loss is zero.

Early stopping means halting the optimization procedure before reaching the local minimum. Early stopping criteria is used on a validation set when overfitting begins to occur. It should be noted that the true loss function instead of the surrogate loss function is used to

check the criteria for early stopping. This is one of the key differences between traditional optimization and optimization used in the training machine learning algorithms.

Stochastic Gradient Descent A common problem with NNs is that large training sets are necessary for good generalization while at the same time training large datasets is also computationally more expensive. Before deep learning, kernel learning algorithms were the primary choice for nonlinear models. Many of these algorithms require constructing a kernel matrix of size $N \times N$, where N is the size of the dataset. This requires a computational cost of $\mathcal{O}(N^2)$, which is clearly undesirable for a dataset with millions of samples. The success of deep learning algorithms lies in finding a scalable way of stochastic gradient descent, which is an extension of the gradient descent algorithm [Cau47], in learning nonlinear models with large datasets.

The gradient descent algorithm uses the gradient of the loss $\mathcal{L}$ with respect to parameters $\boldsymbol{\theta}$ and reduce the loss function by moving $\boldsymbol{\theta}$ in small steps with opposite sign of the derivative. This method may not be guaranteed to arrive at even a local minimum in a reasonable amount of time, but it often finds a low value of the cost function quickly enough to be useful. The loss function used by NNs is often decomposed as a sum of some per-example loss functions over all training examples. For example, the maximum likelihood loss function can be written as

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = E_{\mathbf{x},\mathbf{y}\sim p_{data}(\mathbf{x})} p_{model}(\mathbf{x}|\boldsymbol{\theta}). \tag{4.21}$$

The expectation is approximated using the mean loss over all data samples, which is the unbiased estimator of the expectation as follow:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{x}_i, y_i, \boldsymbol{\theta}). \tag{4.22}$$

This means the gradient descent requires performing the operation

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}_i, y_i, \boldsymbol{\theta}), \tag{4.23}$$

Algorithm 4.1 `Stochastic gradient descent at iteration` k

Require: : Learning rate ϵ_k.
Require: : Initial parameter value $\boldsymbol{\theta}$.
1: **while** stopping criteria not met **do**
2: Sample a minibatch of m' samples from the training set along with their corresponding target
3: Compute a gradient estimate $\nabla_{\boldsymbol{\theta}} \mathcal{L}'(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$
4: Perform parameter update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \nabla_{\boldsymbol{\theta}} \mathcal{L}'(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$
5: **end while**

with computation cost as $\mathcal{O}(N)$. Although this is more efficient than kernel methods, it is still not realizable with millions or billions of data samples. Stochastic gradient descent provides a solution to this problem by computing the loss function gradient only with a small number $N' \ll N$ of examples sampled randomly from the training dataset. Therefore, this method is also known as the minibatch stochastic gradsient method. This is possible as stochastic gradient descent is a procedure to compute an unbiased estimate of the gradient, which can be approximated with a small set of samples. The reason behind sampling randomly is that samples need to be independent of each other for computing the unbiased estimate of the expected gradient. The estimate of the gradient becomes

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}'(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{1}{N'} \sum_{i=1}^{N'} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}_i, y_i, \boldsymbol{\theta}) \tag{4.24}$$

The stochastic gradient descent algorithm then follows the estimated gradient downhill to update the parameters $\boldsymbol{\theta}$ such that

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \nabla_{\boldsymbol{\theta}} \mathcal{L}'(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}), \tag{4.25}$$

where ϵ_k is the learning rate, a positive scalar determining the size of the step. This parameter is gradually decreased over iterations as it approaches a local or global minima. Thus, its value depends on the number of iteration k. The most important property of minibatch stochastic gradient descent is that the computation time per parameter update is independent of the size of the training dataset. Alg. 4.1 describes the steps in the k^{th} iteration of the stochastic gradient descent.

Stochastic Gradient Descent with Momentum Although SGD is a very successful optimization strategy, learning with it is sometimes slow. This is because it calculates noisy gradients of the loss function as they are just estimated gradients on a small set of random samples and not true gradients. Here, the exponentially weighted average method provides a better estimate than noisy gradients calculated by SGD. Therefore, the method of momentum [Pol64] is designed to accelerate learning with noisy gradients. This algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. This is also helpful to accelerate learning in cases where high curvature gradients or consistent gradients with small steps are present.

The name momentum derives from a physical analogy, in which the negative gradient is a force moving a particle through parameter space, according to the Newton's laws of motion. Momentum in physics is mass times velocity. In the momentum learning algorithm, we assume unit mass, so the velocity vector $\boldsymbol{v}$ may also be regarded as the momentum which determines the direction and speed at which the parameters move through parameter space. The velocity is set to an exponentially decaying average of the negative gradient whose contributions is defined using the hyperparameter $\alpha_M \in [0, 1)$. The update rule changes to:

$$\boldsymbol{v} \leftarrow \alpha_M \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}'(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}), \tag{4.26}$$

Algorithm 4.2 `Stochastic gradient descent with Momentum at iteration` k

Require: : Learning rate ϵ_k.
Require: : Initial parameter value $\boldsymbol{\theta}$.
1: **while** stopping criteria not met **do**
2: Sample a minibatch of m' samples from the training set along with their corresponding target
3: Compute a gradient estimate $\nabla_{\boldsymbol{\theta}} \mathcal{L}'(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$
4: Compute a velocity update $\boldsymbol{v} \leftarrow \alpha_M \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}'(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$
5: Perform parameter update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
6: **end while**

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{4.27}$$

There are many other methods of gradient descent optimization such as ADAgrad, RMSProp, ADAM, second order Newton's method, which have additional characteristics to avoid the problem of ending in suboptimal local minima and choosing the correct learning rate.

Architecture of NNs

Parameter Initialization Strategies Training algorithms for deep learning models are usually iterative and thus require the user to specify some initial point from which to begin the iterations. The initial point can determine how quickly learning converges and whether it converges. Designing improved initialization strategies is a difficult task because NN optimization is not yet completely understood. Perhaps the only property known with complete certainty is that the initial parameters need to "break symmetry" between different units. If two hidden units with the same activation function are connected to the same input, then these units must have different initial parameters. If they have the same initial parameters, then a deterministic learning algorithm applied to a deterministic cost and model will regularly update both of these units in the same way. The goal of having each unit compute a different function motivates the random initialization of the parameters. The weights of the network are typically assigned randomly from a Gaussian or uniform distribution and biases for each unit are chosen constants.

Thus, initializing parameters with $\boldsymbol{\theta}_0$ can be considered as imposing a Gaussian or uniform prior with mean $\boldsymbol{\theta}_0$. If $\boldsymbol{\theta}_0$ is close to zero, it suggests that the units more likely do not interact with each other than they interact. They will interact with each other only if the loss function indicates a strong interaction between them during the backpropagation procedure. On the other hand, initializing $\boldsymbol{\theta}_0$ with large values suggests strong prior interaction between units, which is undesirable.

There are some heuristics available for initializing the weights. Most commonly used heuristic for weights of fully-connected layer is *normalized initialization* [GB10]. The weights are

initialized for a fully-connected layer with m inputs and n outputs such that

$$\boldsymbol{W} \sim \mathcal{U}\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right). \tag{4.28}$$

Activation Functions The activation function σ is used in NNs to add nonlinearity to the model. Apart from being nonlinear, these activation functions should be derivable, so that the error can be backpropagated through them using the chain rule during training the network with the backpropagation procedure. Hidden units in the network accept a vector of inputs $\boldsymbol{x}$, compute an affine transformation $\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, and then a nonlinear element-wise activation function $\boldsymbol{\sigma}(\boldsymbol{h})$ is applied. Some of the most popular activation functions are shown in Fig. 4.5 and are described below:

- Logistic Sigmoid: The sigmoid activation function is $\sigma(h) = \frac{1}{1+e^{-h}}$. These units are only sensitive to their input when h is near zero. Otherwise, they saturate to a high positive and high negative value of h. This characteristic is called *vanishing gradient problem*. Therefore, they are not much popular nowadays for hidden units in a NN, but they are used for output units to predict the probabilities for classes, especially in binary output problems as they output values between 0 and 1.
- Hyperbolic tangent: Hyperbolic tangent function is $\sigma(h) = tanh(h) = \frac{2}{1+e^{-2h}}$. They look very similar to the sigmoid function. In fact, they are scaled sigmoid functions such that $tanh(h) = 2sigmoid(2h) - 1$. The difference is they have a larger gradient than sigmoid function and the range of the output is $[-1, 1]$. Like sigmoidal units, hyperbolic tangent functions also suffer from vanishing gradient problem.
- Rectified linear units: Rectified linear units (ReLU) use the activation function $\sigma(h) = max(0, h)$. These units output zero for negative input while it works as a linear function for positive inputs. Thus the derivatives through a rectified linear unit remain large whenever the unit is active. The side effect of this is that these units sometimes suffer from exploding gradient problem in which the gradient becomes too large.
- Softmax: Softmax functions are most often used as the output of a classifier, to represent the probability distribution over multiple different classes. It requires not only that each element of output be between 0 and 1, but also that the entire vector sums to 1 so that it represents a valid probability distribution. The softmax function can then exponentiate and normalize h to obtain the desired output. Formally, it is defined as $\sigma(h_i) = \frac{e^{h_i}}{\sum_j e^{h_j}}$. Softmax can be seen as a generalization of sigmoid function for multiple classes.

4.2.2 Convolutional Neural Networks

Convolutional Neural Networks (ConvNets) are NNs that use convolution in place of general matrix multiplication in at least one of their layers. Mainly, three types of layers are stacked

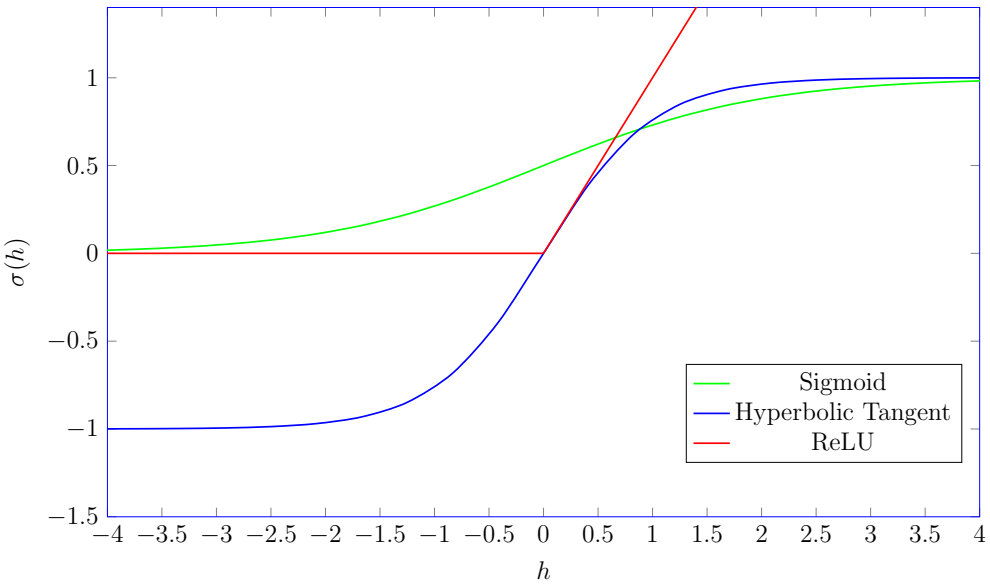


Figure 4.5: Common Activation Functions.

one after another to build ConvNets: convolution layer, pooling layer, and fully-connected layer.

The *convolution layer* implements two important ideas to improve a machine learning system: *sparse interaction* and *parameter sharing* [GBC16]. In traditional NN layers, the interaction between each output and input unit is described by a separate parameter. The convolution layer has *sparse interaction* by making the convolution kernels ${}_k\boldsymbol{W}$ smaller than the input $\boldsymbol{U}$. This results into fewer number of parameters and operations to compute the output, decreasing memory and time requirements of the model. Further, every kernel is slid over the entire input dimension and convoluted. This *parameter sharing* approach leads to further reduction in parameters of the model. The convolution layer is usually followed by a nonlinear activation function $\boldsymbol{\sigma}$ such as sigmoid, rectified linear unit, etc. Thus the element at position x, y of the k^{th} feature map ${}_k\boldsymbol{V} \in \mathbb{R}^{v_1 \times v_2}$, having height v_1, width v_2, with the filter ${}_k\boldsymbol{W} \in \mathbb{R}^{k_1 \times k_2}$, having height $k_1 < v_1$, width $k_2 < v_2$, is

$$ {}_kV^{xy} = \sigma\left({}_kb + \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} {}_kW_{ij}U_{ij}^{xy}\right), \tag{4.29} $$

where U_{ij}^{xy} is the input at position $(x + i\Delta x, y + j\Delta y)$. The input for the convolutional layer, that is not the first layer in the NN, is the output of the previous layer. The point to note here is that the the filter $\boldsymbol{W}$ is flipped horizontally and vertically first and then above equation is calculated as per the definition of the convolution [Ahn]. A visualization of this convolution operation is shown in the Fig. 4.6.

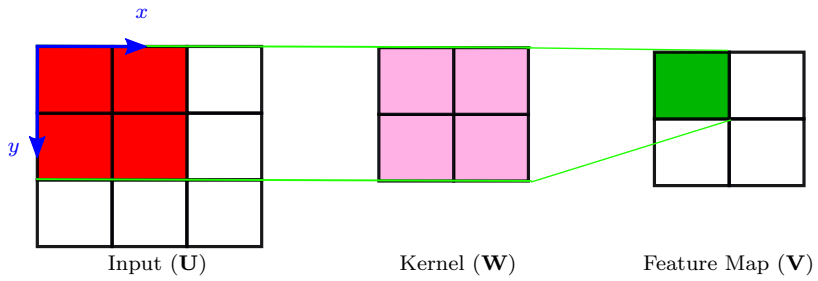


Figure 4.6: 2D Convolution Operation.

The *pooling layer* is generally inserted between successive convolution layers. It converts the convolutional layer output into a number of slices of dimensions $k'_1 \times k'_2$ and finds a single summary representation for each slice. Thus, it reduces the amount of the parameters and computation in the network. The most commonly used pooling functions are max-pooling and average-pooling, which are shown in the Fig. 4.7. The operations performed to calculate an element in the output of average-pooling is

$$ {}_kP^{xy} = \frac{\sum_{i=0}^{k'_1-1} \sum_{j=0}^{k'_2-1} {}_kV_{ij}^{xy}}{k'_1 k'_2}, \tag{4.30} $$

where ${}_kV_{ij}^{xy}$ is the input at position $(x+i\Delta x, y+j\Delta y)$. Similarly, an element in the output of max-pooling layer is

$$ {}_kP^{xy} = max\left({}_kV^{x+0,y+0}, {}_kV^{x+1,y+0}, \ldots, {}_kV^{x+k'_1,y+k'_2}\right). \tag{4.31} $$

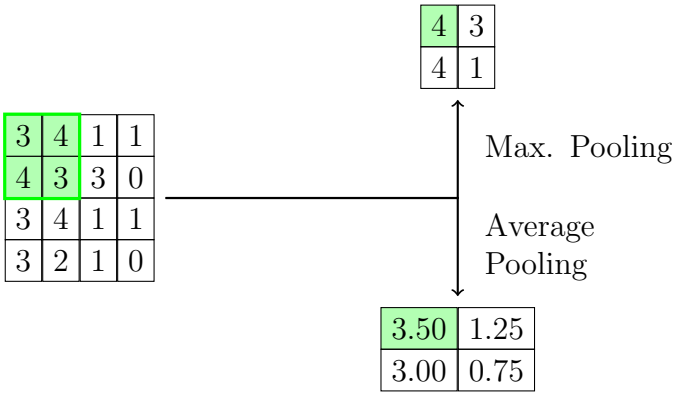


Figure 4.7: Pooling Operations.

After several convolution and pooling layers a fully-connected layer follows. The output layer has generally one neuron per class in the classification task. A softmax function is used so that each output neuron represents the posterior class probability. The backpropagation algorithm for ConvNets to learn the parameters is illustrated in Appendix 8.2.5.

4.2.3 3D ConvNets

In order to extract spatiotemporal features from $\mathbf{M}$, a 3D convolution neural network (3D-ConvNet) [JXYY10] is used. It performs a 3D convolution in convolutional layers and then an additive bias b is applied and the result is passed through an activation function σ. As shown in Fig. 4.8, the value of unit ${}_kV^{xyt}$ in a k^{th} feature map at position (x, y) and time t is obtained by

$$ {}_kV^{xyt} = \sigma\left({}_kb + \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1}\sum_{n=0}^{k_3-1} {}_kW_{ijn}U_{ijn}^{xyt}\right), \tag{4.32} $$

where k_1, k_2, and k_3 are the dimensions of the kernel ${}_k\mathbf{W}$ and U_{ijn}^{xyt} is the input at position $(x + i\Delta x, y + j\Delta y, t + n\Delta t)$. It also performs a 3D pooling instead of 2D pooling.

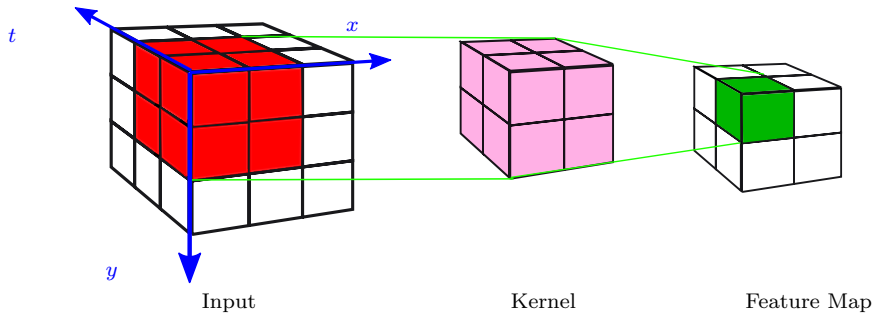


Figure 4.8: 3D Convolution Operation.

4.2.4 Recurrent Neural Network

The ConvNets learn the spatial patterns from inputs having a grid-like structure while the recurrent neural networks (RNN) make use of the sequential structure of the input. In a traditional NN, we assume that all inputs (and outputs) are independent of each other. This is primarily a bad idea for applications where the next output depends on the previous computations, such as in predicting the next word in a sentence. RNNs are recurrent because they perform the same computation for every element of a sequence, with the output being dependent on the previous computations. RNNs can be thought to have a "memory", which captures information about what has been calculated so far based on which it makes the inference. Fig. 4.9 shows the basic structure of a RNN for a sequence with N sequential steps. The representation on the right-hand side shows the internal loops indicating the "memory" component in RNN. It can be unrolled to get the full structure of the network on the right-hand side.

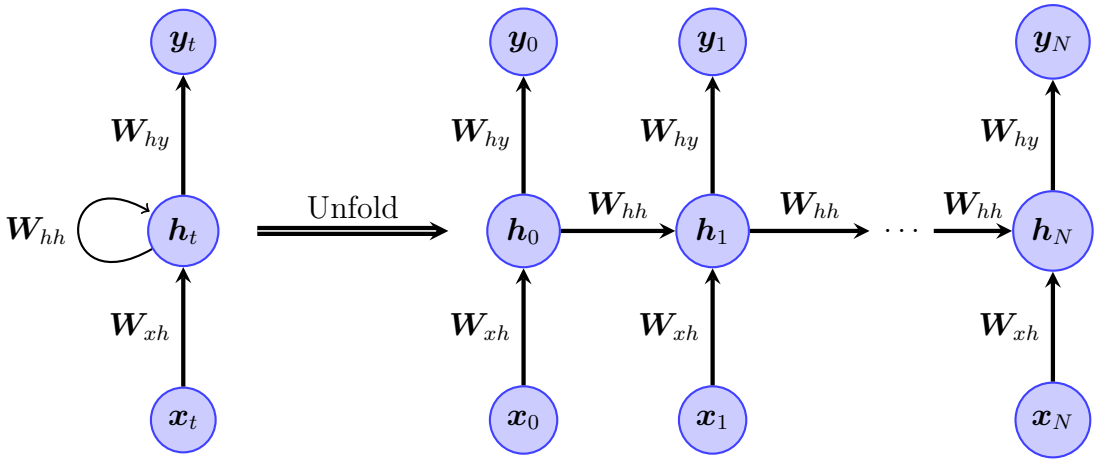


Figure 4.9: The Structure of Recurrent Neural Network with N Sequential Steps.

There are some important parameters from Fig. 4.9, which need to be described to understand RNN better. The vector $\boldsymbol{x}_t$ is the input at step t. Here, t can be a time step or just the sequence step. The vector $\boldsymbol{h}_t$ is the hidden or latent state calculated based on previous latent states and input at the current step such that

$$\boldsymbol{h}_t = \boldsymbol{\sigma}(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}), \tag{4.33}$$

where $\boldsymbol{\sigma}$ is result of application of a nonlinear activation function. The vector $\boldsymbol{h}_{t-1}$, required to calculate the first hidden state, is usually initialized to zero. This component can be thought of as the "memory" component of the network as it captures information about what happened in all the previous steps. Finally, $\boldsymbol{y}_t$ is the output at step t. It is a vector of probabilities over the labels. Therefore, it is calculated as

$$\boldsymbol{y}_t = softmax(\boldsymbol{W}_{hy}\boldsymbol{h}_t). \tag{4.34}$$

The noticeable thing about RNN is that it shares parameters across all steps, unlike other NNs that use different parameters in each layer. The assumption is that the network is performing the same task at each step irrespective of the inputs. It greatly reduces the number of parameters. These parameters are also learned by the backpropagation procedure. As the parameters are shared, the gradient at each output depends on the calculations of previous steps as well. Therefore, the gradient is summed up over all previous steps. This procedure is called *Backpropagation Through Time*. It looks like RNNs are capable of using the information in arbitrarily long sequences, but in practice, they suffer from problems like vanishing and exploding gradient. The extensions of RNN, such as LSTM [HS97] and GRU [CvMG+14], which handle these problems efficiently, are more popular for problems involving long sequences.

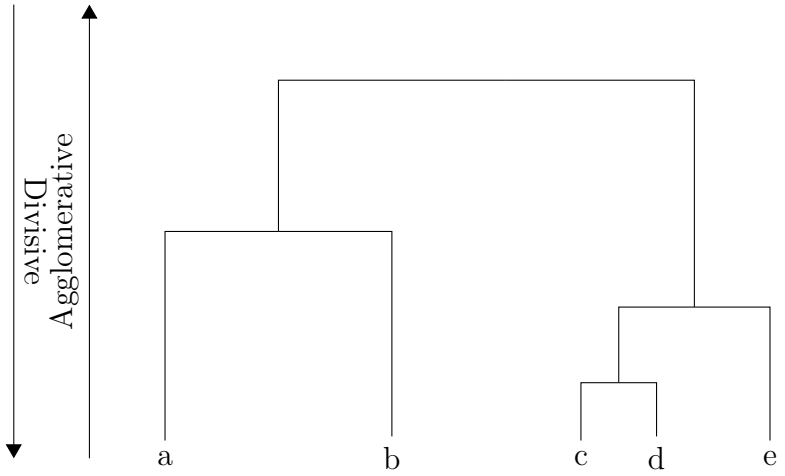


Figure 4.10: Hierarchical Clustering Types.

4.3 Unsupervised Learning Algorithms

4.3.1 Clustering Algorithms

Clustering is classifying samples into groups, also called clusters, such that the samples in one group are similar to each other and dissimilar to the samples from other groups. A distance metric measures the similarity or dissimilarity. The most commonly used distance metric is the Euclidean distance. Typically, clustering algorithms are divided into two groups: hierarchical and nonhierarchical.

Nonhierarchical clustering methods such as k-means clustering methods as they iteratively calculate cluster centroids with samples are added or subtracted from the cluster till there is no reassignment. *Hierarchical clustering* is a method of cluster analysis which groups data over a variety of scales by creating a cluster tree called a dendrogram. The dendrogram is a multilevel hierarchy in which clusters at one level are either combined or further split into the next level. Based on this property, the hierarchical clustering is classified into two approaches:

- Agglomerative: In this type, each data sample is assigned to an individual cluster on the first level of the tree and at each level, the closest pair of clusters are merged. Therefore, this requires a definition of cluster proximity.
- Divisive: It starts with all the objects grouped in a single cluster. Clusters are divided or split until each object is in a separate cluster.

Fig. 4.10 shows the direction of both agglomerative and divisive hierarchical clustering and different levels of the dendrogram with five samples.

Agglomerative methods are more commonly used than divisive methods. They consist of linkage methods, variance methods, and centroid methods based on the definition of

cluster proximity. The linkage method is further subdivided into a single, complete and average linkage method, which measures minimum, maximum and average distance between samples of the cluster, respectively. Variance methods generate clusters to minimize the within-cluster variance. In centroid methods, the distance between two clusters is the distance between their centroids.

4.3.2 Autoencoder

An autoencoder is a NN that consists of two parts, namely encoder and decoder. It is trained to approximate its input $\boldsymbol{x}$ to the output. The encoder $f_{\mathbf{x}|\mathbf{z}}$ part converts the input $\boldsymbol{x}$ to the latent layer $\boldsymbol{z}$, while the decoder tries to reconstruct the input $\boldsymbol{x}$ from the latent layer $\boldsymbol{z}$. The model is trained with the gradient descent method using the reconstruction error of the input as the loss function. The commonly used loss functions for reconstruction error are the L_1 and L_2 norms. The size of the hidden layer $\boldsymbol{z}$ is smaller than the size of the input $\boldsymbol{x}$ to to obtain a lower dimensional representation of the input. This forces the model to learn only the most important features. Traditionally, autoencoders were used for dimensionality reduction, but modern autoencoders have generalized the idea of an encoder and a decoder beyond deterministic functions to stochastic mappings.

Figure 4.11: The Architecture of an Autoencoder.

4.3.3 Variational Autoencoder

A variational autoencoder (VAE) is a type of generative model which deals with learning the data distribution $p_{\mathbf{x}}(\boldsymbol{x})$ from which new data samples of the input $\boldsymbol{x}$ can be generated. VAE makes the assumption that the input data is generated by a two step random process using latent variables $\boldsymbol{z}$, whose values can be sampled from a probability distribution $p_{\mathbf{z}}(\boldsymbol{z})$. The generative model $g(\boldsymbol{z}, \boldsymbol{\theta})$, parametrized by $\boldsymbol{\theta}$, represents a parametrization of the likelihood function $p_{\mathbf{x}|\mathbf{z}}(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta})$ such that

$$p_{\mathbf{x}}(\boldsymbol{x}) = \int p_{\mathbf{x}|\mathbf{z}}(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta}) p_{\mathbf{z}} d\boldsymbol{z}, \tag{4.35}$$

where the prior $p_{\mathbf{z}}$ can be a simple distributions like the standard normal distribution or the Bernoulli distribution. The goal is to find the parameters $\boldsymbol{\theta}$ to maximize to maximize the probability for observing the data in the training set. Eq. 4.35 can be approximated with the samples of $\mathbf{z}$ such that

$$p_{\mathbf{x}}(\boldsymbol{x}) = E_{\mathbf{z}}\left[p_{\mathbf{x}|\mathbf{z}}(\boldsymbol{x}|\mathbf{z})\right]. \tag{4.36}$$

This equation is not practically computable as many samples of $\mathbf{z}$ are required and for most of the samples $p_{\mathbf{x}|\mathbf{z}}(\boldsymbol{x}|\mathbf{z})$ would be negligible, i. e.,$p_{\mathbf{x}|\mathbf{z}}(\boldsymbol{x}|\mathbf{z}) \approx 0$. Therefore, the distribution

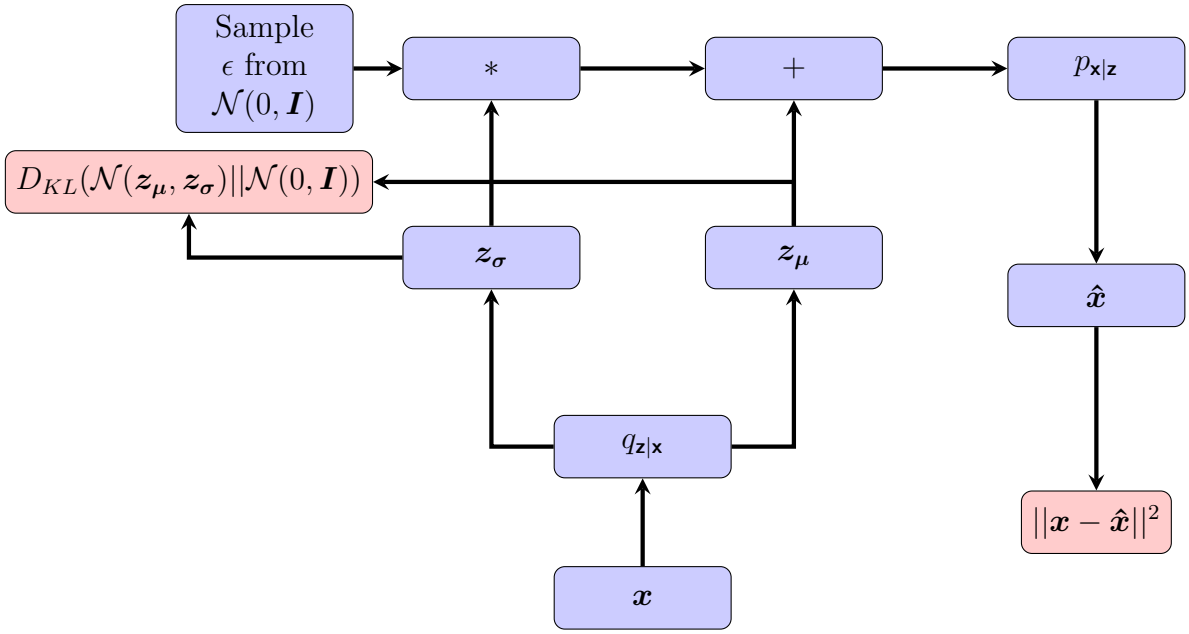


Figure 4.12: Structure of Variational Autoencoder.

$p_\mathbf{z}$ is restricted to a family of simpler distributions in order to approximate it to $p_{\mathbf{z}|\mathbf{x}}$. As this is unknown as well it is learned with model $q_{\mathbf{z}|\mathbf{x}}$. This can be achieved by minimizing the Kullback-Leiber divergence (KL divergence)

$$D_{KL}(q_{\mathbf{z}|\mathbf{x}}||p_\mathbf{z}) = E_{\mathbf{z}\sim q_{\mathbf{z}|\mathbf{x}}}\left[\log \frac{q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}|\mathbf{x})}{p_\mathbf{z}(\mathbf{z}|\mathbf{x})}\right]. \tag{4.37}$$

Applying Bayes rule to Eq. 4.37, the equation can be rewritten as

$$D_{KL}(q_{\mathbf{z}|\mathbf{x}}||p_\mathbf{z}) = E_{\mathbf{z}\sim q_{\mathbf{z}|\mathbf{x}}}\left[\log \frac{q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}|\boldsymbol{x})}{p_{\mathbf{x}|\mathbf{z}}(\boldsymbol{x}|\mathbf{z})} \frac{p_\mathbf{x}(\boldsymbol{x})}{p_\mathbf{z}(\mathbf{z})}\right]. \tag{4.38}$$

Consequently, reformulating the above equation

$$\log p_\mathbf{x} - D_{KL}(q_{\mathbf{z}|\mathbf{x}}||p_{\mathbf{z}|\mathbf{x}}) = E_{\mathbf{z}\sim q_{\mathbf{z}|\mathbf{x}}}\left[\log p_{\mathbf{x}|\mathbf{z}}(\boldsymbol{x}|\mathbf{z})\right] - D_{KL}(q_{\mathbf{x}|\mathbf{z}}||p_\mathbf{z}). \tag{4.39}$$

The second term on the right hand side of the above equation, i. e., the KL divergence between the $q_{\mathbf{z}|\mathbf{x}}$ and $p_{\mathbf{z}|\mathbf{x}}$ is intractable as both the distributions are unknown. However, it is always greater or equal to 0 as per the definition of KL divergence. This means that

$$E_{\mathbf{z}\sim q_{\mathbf{z}|\mathbf{x}}}\left[\log p_{\mathbf{x}|\mathbf{z}}(\boldsymbol{x}|\mathbf{z})\right] - D_{KL}(q_{\mathbf{x}|\mathbf{z}}||p_\mathbf{z}). \tag{4.40}$$

is a *lower bound* for $\log p_\mathbf{x}$ which is tighter the more the model distribution $q_{\mathbf{z}|\mathbf{x}}$ approaches the true, but unknown, distribution $p_{\mathbf{z}|\mathbf{x}}$. The problem formulation in Eq. 4.35 is changed

to maximizing the lower bound in Eq. 4.40. This creates two loss functions. First, the reconstruction error as in simple autoencoders represented by the first term in Eq. 4.40. By maximizing the likelihood, $q_{\mathbf{z}|\mathbf{x}}$ is forced to produce $\boldsymbol{z}$ that $p_{\mathbf{x}|\mathbf{z}}$ can reliably decode to reconstruct the input. The second loss function is the KL divergence between $q_{\mathbf{z}|\mathbf{x}}$ and $p_{\mathbf{z}}$, where $p_{\mathbf{z}}$ is usually defined as the normal distribution $\mathcal{N}(0, \boldsymbol{I})$. One crucial problem is training this autoencoder with the backpropagation algorithm because of the presence of sampling which is not differentiable. This problem is solved by the reparameterization trick [KW13]. Instead of sampling $\boldsymbol{z} \sim q_{\mathbf{z}|\mathbf{x}}$, a sample $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$ is chosen and $\boldsymbol{z}$ is calculated as

$$\boldsymbol{z} = \boldsymbol{z}_{\boldsymbol{\mu}} + \epsilon \boldsymbol{z}_{\boldsymbol{\sigma}}, \tag{4.41}$$

where $\boldsymbol{z}_{\boldsymbol{\mu}}$ and $\boldsymbol{z}_{\boldsymbol{\sigma}}$ are considered to be vectors generated from $q_{\mathbf{z}|\mathbf{x}}$. The overall architecture of VAE including the reparameterization trick and two loss function is shown in Fig. 4.12.

Summary

In the first part of this chapter, basic concepts of machine learning are presented. In later parts of this chapter, different supervised and unsupervised learning algorithms relevant for this work are described.

Chapter 5

Hybrid Machine Learning Methods for Trajectory Planning

The proposed model-based trajectory planning algorithms in Chapter 3, namely ARRT and ARRT+, are potent algorithms to find safe trajectories in complex, critical traffic-scenarios. They are probabilistically complete algorithms, which means they will always find a solution, if it exists, given infinite time. However, these algorithms require high computational resources to run in real-time. It is because of the complexity of the underlying problem, usage of a highly accurate vehicle dynamic model and the uncertainty in the nature of the algorithms due to random sampling. These algorithms may not be able to find a safe trajectory in real-time even if the traffic-scenarios is not complicated because the random sampling can generate undesirable samples which hamper the fast convergence. Therefore, these algorithms are not directly realisable in a vehicle where only limited onboard computational resources are available.

Another inescapable feature of randomized algorithms is their lack of repeatability as no two runs will execute identically. There is no guarantee that these algorithms will converge again, in the same way in which they converged previously, given a time constraint. In trajectory planning, this has both positive and negative implications. The positive thing is that sometimes the randomized algorithm will be "lucky" and solve a complex problem very quickly. This might not be the case for a deterministic algorithm. If a deterministic algorithm performs poorly once, it will always perform poorly for that problem. On the other hand, the lack of repeatability caused by randomization can easily hide how well the algorithm performs and can cause flaws to be overlooked. This makes the testing and the validation of the randomised algorithms harder. With deterministic algorithms, a single execution is sufficient to find flaws. Hence, working with deterministic algorithms can result in greater carefulness in both algorithm design and implementation.

Machine learning methods, that are briefly explained in Chapter 4, are considered to be *Black-Box* methods as they are purely based on data. Therefore, machine learning have

not found their way in safety-critical applications like safe trajectory planning where the precise understanding of the models, those defining the control inputs for actuators, is required.

This work proposes a combination of machine learning methods with sampling-based algorithms to exploit the advantages of both types of methods. The basic idea of the combination is to assist sampling-based methods with machine learning algorithms, such that a suitable prior for sampling is generated by machine learning. With a suitable prior the convergence of the sampling-based algorithms can be speeded up. It is acceptable to use the machine learning methods in such a manner for safety-critical applications because the final solution is generated by a model-based algorithm that uses sampling and vehicle dynamic constraints.

The overview of this chapter is as follows: Section 5.1 briefly describes the planned combination of the machine learning algorithms with the sampling-based ARRT and ARRT+ algorithms. It is followed by the literature research of machine learning applications for the trajectory planning algorithms in Section 5.2. Section 5.3 explains the hybrid machine learning algorithms for the sampling-based trajectory planning algorithms in detail, while Section 5.4 illustrates the hybrid machine learning algorithms for a deterministic trajectory planning algorithm. The simulation results for the comparison of the trajectory planning algorithms in different critical traffic-scenarios are provided in these sections. Finally, Section 5.4.6 presents the similarities and dissimilarities between the proposed hybrid machine learning algorithms and reinforcement learning algorithms.

5.1 Proposed Combination of Sampling-Based and Machine Learning Methods

With the above mentioned issues related to complete random sampling, it is clear that a non-uniform or biased sampling can be more beneficial than uniform sampling. The primary motivation for non-uniform sampling is simple. If it is possible to determine that certain regions of the configuration space are more important than others, then one would like to be able to sample these at a higher frequency, as efficiently as possible. However, it is very challenging to find how to explore the configuration space with non-uniform sampling. In general, there are two approaches to non-uniform sampling: *importance sampling* and *adaptive sampling*. Importance sampling is based on the prior belief that solutions will be found more quickly by concentrating on sampling in certain areas of configuration space. In adaptive sampling, the distribution from which new samples are drawn is modified based on information gained from previous samples. This thesis adopts the importance sampling technique.

As explained earlier, there are disadvantages and advantages associated with both model-based and machine learning methods. Therefore, *hybrid machine learning methods*, which are a combination of model-based and machine learning methods, are proposed in this work

for safety-critical applications like safe trajectory planning. These methods open a new way to simultaneously exploit the advantages of machine learning methods and eliminate their disadvantages by combining them with physical models. Therefore, they can be used in safety-critical applications like safe trajectory planning. The basic idea of the proposed combination is to find an approximate solution by machine learning methods and use this predicted solution as a reference for the fast convergence of model-based sampling algorithms. Specifically, for the given task of vehicle safe trajectory planning, the concept of importance sampling is used. Machine learning algorithms generate a prior belief to change the sampling strategy of the ARRT and ARRT+ algorithms.

The effect of non-uniform sampling on the performance of sampling-based motion planning algorithms is still an open research question. The experimental results presented in [LL03] and [GO07] shows that there is no single sampling strategy that outperforms others in every scenario. Therefore, a general rule-based sampling strategy applicable to all types of scenarios is difficult to find. It endorses further the use of machine learning algorithms for deducing the non-uniform sampling strategy.

Machine learning algorithms generate a bias with the ARRT and ARRT+ algorithms in different ways. The ARRT algorithm uses L longitudinal acceleration profiles $\boldsymbol{a}_{\mathbf{x}}^{i}$, where $i = 1, \ldots, L$, sequentially to find multiple safe trajectories. The increase in the number of acceleration profiles raises the chance of finding safe trajectories in a complex multi-object dynamic traffic-scenario, but it also raises the computation time. Therefore, a machine learning algorithm is proposed to predict only the best l out of L acceleration profiles for a traffic-scenario, and use them with the ARRT algorithm to find safe trajectories. This combination of the ARRT algorithm with a machine learning algorithm is named as the *Hybrid Augmented CL-RRT* (HARRT) algorithm.

The ARRT+ algorithm does not use fixed acceleration profiles. Therefore, in order to generate a training dataset for machine learning, the safe trajectories from the ARRT+ algorithm in multiple critical traffic scenarios are represented by templates and labels are assigned to these templates. The trained machine learning algorithm then predicts these template trajectories and biases the sampling of the ARRT+ algorithm around those trajectories. This hybrid machine learning algorithm is named as the *Hybrid Augmented CL-RRT+* (HARRT+) algorithm. The advantage of these hybrid machine learning approaches is that the model-based algorithms still find safe trajectories with a reduction in the convergence time. Section 5.3 describes the HARRT and HARRT+ algorithm in detail.

Even after the combination with machine learning algorithms, the issue of lack of repeatability with the ARRT and ARRT+ algorithm is still present because of the randomness in their nature. An alternative for this is the combination of machine learning algorithms with deterministic algorithms for safe trajectory planning. The idea of this combination is the same. The machine learning algorithm finds an approximate solution and then a deterministic algorithm adapts this solution as per the defined constraints. Specifically, the machine learning algorithms generate a reference trajectory, which by taking as an initial solution, an optimization-based deterministic algorithm modifies this trajectory. This al-

gorithm is named as the *Generative Algorithm for Trajectory Exploration* (GATE). Section 5.4 illustrates this algorithm and further its combination with the ARRT+ algorithm.

5.2 Literature Research on Machine Learning for Trajectory Planning

The use of machine learning algorithms for trajectory planning is not a widely researched area. In [BIS09], many machine learning algorithms for the subtasks of trajectory planning are mentioned. However, they are not safety-critical tasks.

The RRT variants which incorporate machine learning algorithms and their limitations are described in Section 3.1.5. There are some hybrid machine learning algorithms proposed for board games. AlphaGo [SHM+16] and ExIT [ATB17] are two examples of guided tree search algorithms with neural networks for the board games Go and Hex, respectively. Although these approaches are considered breakthroughs in artificial intelligence, they are still narrow approaches suitable only for games and not real-world applications because of various reasons. First of all, these algorithms are limited to discrete state-spaces and action-spaces, which is mostly not valid for real-world robotic applications. Also, these games involve a fixed number of players, most commonly two, who act in sequence trying to win by defeating other players without the real-time constraint. On the other hand, the vehicle trajectory planning in critical traffic-scenario consists of an uncertain number of participants in with each participant acts to avoid collision with real-time constraints. There are other differences, such as determinism in the actions of participants and complete observability in the games, which is not available in real-world applications like safe trajectory planning. However, these are not considered in the scope of the approaches proposed in this work. A single deterministic prediction of other traffic-participants and the complete observability of the vehicle surrounding using vehicle sensors is considered. The assumption about a single hypothesis for the prediction of road traffic-participants is justified because the prediction time is small and roads are a structured environment in which the road-participants should follow the rules. The assumption about the complete observability is also justified because of the introduction of modern sensor technologies like V2X and exteroceptive sensors such as lidar, radar, camera, etc. in modern vehicles that provide detailed information of the vehicle surrounding.

The trajectory planning or trajectory prediction problem using machine learning algorithms is not limited to robotic applications. There are other applications for predicting trajectories such as handwriting generation [Gra13] and predicting basketball trajectories [SR16]. These approaches use generative models for trajectory generation. Such algorithms were the motivation for developing a generative model for trajectory planning in this work.

There is simultaneously much interest generated to develop end-to-end learning solution for trajectory planning. The company NVidia proposed such an approach [BYC+17], which uses the image of the road as an input for neural networks and outputs steering angle

values. It tries to make the neural networks interpretable by determining which parts of the image were relevant for decision making through the relevance layerwise propagation. Simultaneously, the application of reinforcement learning algorithms is gaining much interest in trajectory planning. However, these approaches are still in the nascent stage and much research is still needed to be done before they become suitable for high dimensional continuous state-space and action-space problems like vehicle trajectory planning in critical traffic-scenarios.

5.3 Hybrid Machine Learning Algorithms with Sampling-Based Algorithms

This section describes the proposed hybrid machine learning approaches with sampling-based algorithms, i. e., the ARRT and ARRT+ algorithm. The first part of this section explains the details of the design and training details of machine learning algorithms, followed by their actual use in the HARRT and HARRT+ algorithm. Simulation results are presented to compare the HARRT and HARRT+ algorithms to their corresponding sampling-based algorithms based on safety and efficiency in terms of computation time.

5.3.1 Machine Learning Algorithm

Data Generation Procedure

The primary need for any machine learning algorithm is lots of data. The Matlab-based simulation environment described in Section 2.4 is used to design and simulate traffic-scenarios and generate data. Primarily, the data is generated for two-types of challenging traffic-scenarios: 1) curved roads and 2) intersections. Many traffic-scenarios are formed by the combinations of road designs, number and type of road-participants along with the parameters of their initial state such as velocity, position, etc. For the curved roads, the radius of curvature is also changed. An example of a traffic-scenario for both road designs is shown in Fig. 5.1 and Fig. 5.2. The scenarios are simulated using suitable models for road traffic-participants. For the simulation of the EGO vehicle, the nonlinear two-track vehicle dynamic model described in Section 2.1.3, for other vehicles and bicycles the single-track kinematic model explained in Section 2.1.4, while for pedestrians the decoupled lateral and longitudinal dynamics presented in Section 2.1.5 are used. The predictions of the road traffic-participants are found based on the assumptions that vehicles tend to follow the road with a constant velocity and pedestrians travel linearly with a constant velocity. Only critical traffic-scenarios as described in Section 3.3 are considered for the training of the machine learning algorithms. A safe trajectory is planned with the ARRT and ARRT+ algorithms in each scenario. These safe trajectories are used later for the training and the evaluation of the machine learning algorithms.

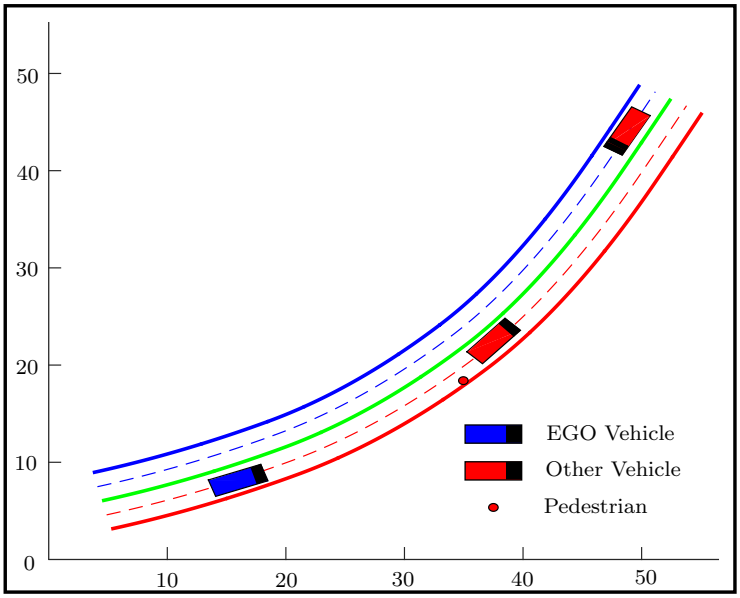


Figure 5.1: An Example of Curved Road Scenario.

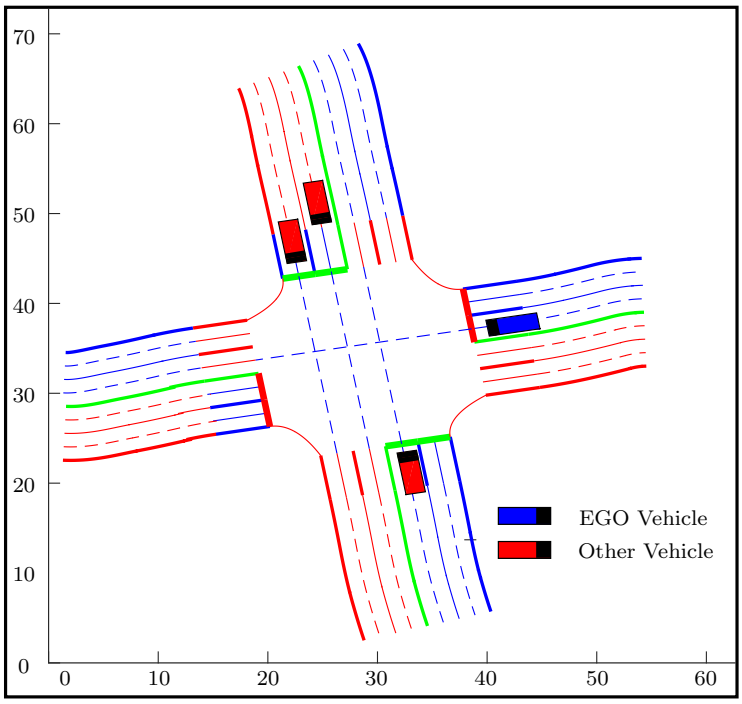


Figure 5.2: An Example of Intersection Scenario.

Feature Space

The construction of the feature vector is essential for achieving good performance with machine learning algorithms. For the vehicle safe trajectory planning, the relevant physical features such as velocity, acceleration, position, etc. of road traffic-participants can be used. However, the disadvantage of using such features is that the size of the feature vector will change with the number of road traffic-participants in the surrounding of the EGO vehicle or with the change in the road infrastructure. A representation of free-space, which is an important piece of information used by every trajectory planning algorithm, is also interesting for the definition of a suitable feature vector. Therefore, a traffic-scenario is converted into a sequence of predicted occupancy grids $\{\mathcal{G}_{t_1}, \ldots, \mathcal{G}_{t_P}\}$ for the prediction interval $[t_1, t_P]$. Such a sequence is denoted as $\mathbf{M}$. An occupancy grid $\mathcal{G}_t$ represents predicted occupancies of road traffic-participants other than the EGO vehicle at prediction time t. A scenario is uniquely defined by a sequence of predicted occupancy grids $\mathbf{M}$ and the vector $\boldsymbol{\eta}$ that represents the EGO vehicle physical parameters such as initial velocity, initial acceleration, etc. The advantage of choosing such feature representation is that the size of the input features remains the same even if the number and type of road traffic-participants or the road infrastructure change.

An example of an occupancy grid $\mathcal{G}_t$ of size 20×40 meters (each cell 1×1 meter) is shown in Fig. 5.3. It shows the EGO vehicle position at the initial time t_0 and predicted positions of other road traffic-participants at time t. The cells of the grid occupied by the predictions of other road traffic-participants and the cells outside of the road surface are assigned a value 1. Otherwise, they are assigned a value 0, indicating they are free.

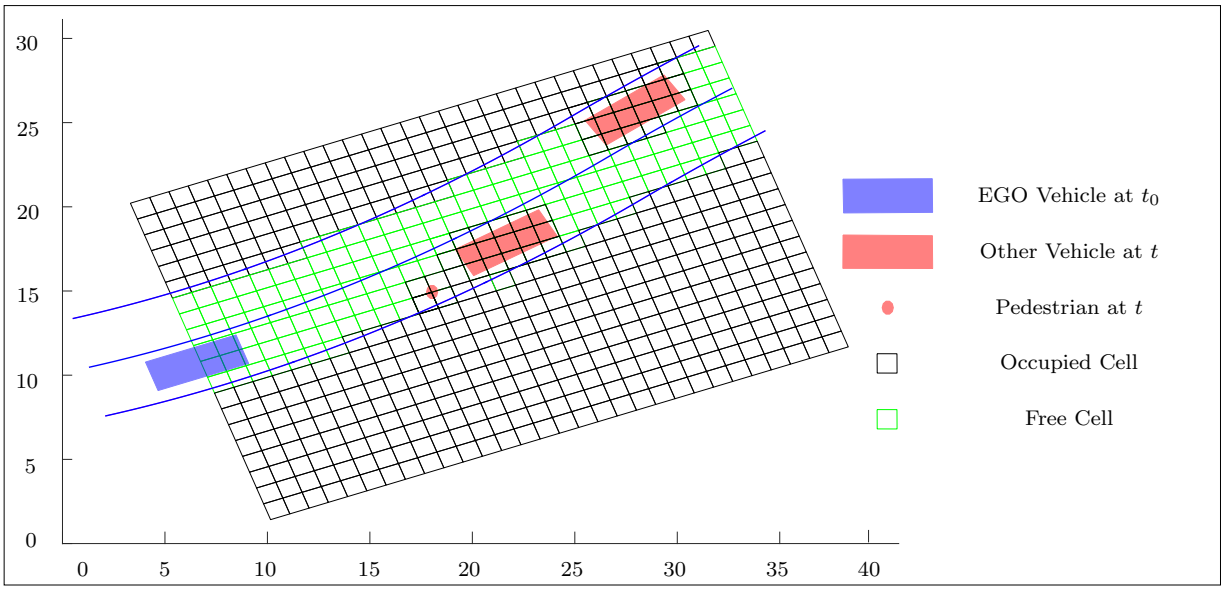


Figure 5.3: Occupncy grid $\mathcal{G}_t$ and EGO Vehicle at Time t_0.

Selection of Machine Learning Algorithms

The sequence of predicted occupancy grids $\mathbf{M}$ has a structure similar to a video. A video is a sequence of two-dimensional images, while $\mathbf{M}$ is a sequence of two-dimensional predicted occupancy grids. Recently, the convolutional neural networks (ConvNets) have shown tremendous results with image and video classification problems. Therefore, ConvNets are chosen as a machine learning algorithm in the HARRT and HARRT+ algorithms. Specifically, 3D-ConvNets are used as the input is three dimensional.

Labeling Procedure

The classification problem requires the data in an input-output pair format. As explained in the previous section, the input features comprise of a stack of predicted occupancy grids $\mathbf{M}$ and the EGO vehicle physical parameters $\boldsymbol{\eta}$. The goal of the machine learning algorithm is to find a function $\boldsymbol{f}_{\boldsymbol{\gamma}}(\mathbf{M}, \boldsymbol{\eta}) \mapsto \hat{\mathbf{y}}$ with learning parameters $\boldsymbol{\gamma}$ to minimize the risk $R(\boldsymbol{f}_{\boldsymbol{\gamma}})$ which is defined as

$$R(\boldsymbol{f}_{\boldsymbol{\gamma}}) = E_{\mathbf{M},\boldsymbol{\eta},\mathbf{y}}\{\mathcal{L}(\mathbf{y}, \boldsymbol{f}_{\boldsymbol{\gamma}}(\mathbf{M}, \boldsymbol{\eta}))\}, \tag{5.1}$$

where $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_k y_k \log(\hat{y}_k)$ represents the cross-entropy loss function, with y_k and $\hat{y}_k$ being the true and estimated posterior probabilities for the k^{th} class. The risk $R(\boldsymbol{f}_{\boldsymbol{\gamma}})$ is replaced by the empirical risk

$$R_{emp}(\boldsymbol{f}_{\boldsymbol{\gamma}}) = \left(\frac{1}{N} \sum_{i=1}^{N} (\mathcal{N}(\mathbf{y}_i, \boldsymbol{f}_{\boldsymbol{\gamma}}(\mathbf{M}_i, \boldsymbol{\theta}_i))) \right), \tag{5.2}$$

where N is the total number of traffic-scenarios in the training data set. Thus, the parameters $\boldsymbol{\gamma}$ of the classifier are found by minimizing $R_{emp}(\boldsymbol{f}_{\boldsymbol{\gamma}})$.

In classification problems, the reference output for 3D-ConvNet is a one-hot encoded vector. It is a vector of size equal to the total number of classes with all of its elements assigned a value 0 except the element at the index k, which is assigned a value of 1 given that the k^{th} label is the true label. These output labels for both the HARRT and HARRT+ algorithms are defined in different ways, such that the labels are suitable for the proposed hybrid algorithms.

The labels for these machine learning algorithms are derived from the safe trajectories found by the model-based algorithms ARRT and ARRT+ in critical traffic-scenarios. Both the ARRT and ARRT+ algorithm can find multiple safe trajectories in a traffic-scenario and they choose the best trajectory as per the steps defined in Section 3.5.8. The ARRT algorithm uses sequentially L longitudinal acceleration profiles $\boldsymbol{a}_{\mathrm{x}}^{l}$, where $l = 1, \ldots, L$, with the random sampling in lateral dynamics to find multiple safe trajectories. The acceleration profile which finds the best trajectory is defined as the label for that particular scenario. The one-hot vector of the size $L+1$ (for L acceleration profiles plus a label for no trajorey found by any acceleration profile) is constructed with the element at k^{th} index assigned value 1 given that with k^{th} acceleration profile, the best trajectory is found.

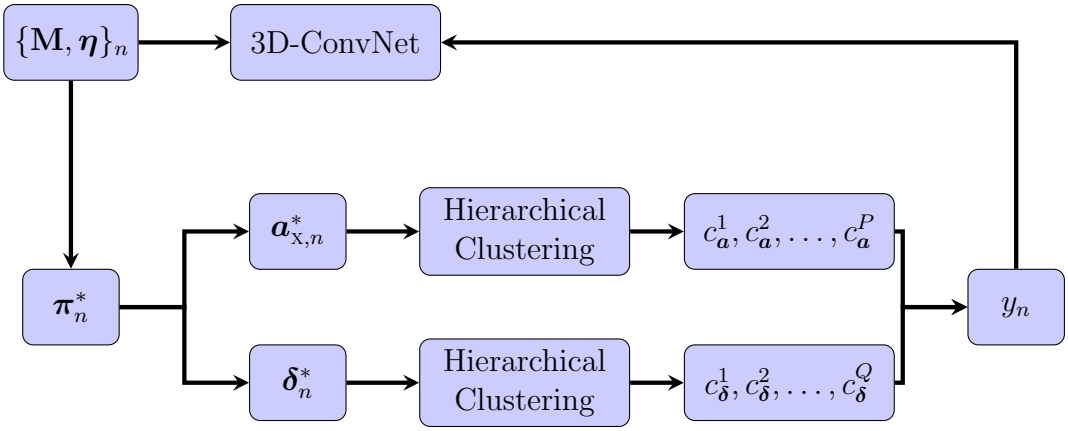


Figure 5.4: Labeling Procedure for HARRT+ Algorithm.

The ARRT+ algorithm also samples in the longitudinal acceleration space and does not use fixed acceleration profiles. Therefore, the labels are generated differently in the case of the HARRT+ algorithm. The labeling procedure for the machine learning algorithm for the HARRT+ algorithm is explained graphically in Fig. 5.4. The best trajectory $\boldsymbol{\pi}_n^*$ is found by the ARRT+ algorithm in all the n^{th} scenario $\{\mathbf{M}, \boldsymbol{\eta}\}_n$, where $n = 1, 2, \ldots, N$. The trajectory $\boldsymbol{\pi}_n^*$ can also be approximated as a combination of the longitudinal acceleration profiles $\boldsymbol{a}_{\mathrm{X},n}^*$ and steering wheel angle profiles $\boldsymbol{\delta}_n^*$. The profile for steering wheel angle is obtained by considering the EGO vehicle as a front-wheel drive vehicle with equal steering wheel angles for front tires and the profiles of longitudinal acceleration is obtained considering equal longitudinal slip values for four tires of the vehicle. Since the machine learning algorithms only predict a trajectory that is used as a reference for biasing the sampling, a more sophisticated modelling for the labelling process is not necessary. Divisive hierarchical clustering is used to form P clusters $c_{\boldsymbol{a}}^1, c_{\boldsymbol{a}}^2, \ldots, c_{\boldsymbol{a}}^P$ of longitudinal acceleration profiles and Q clusters of steering wheel angle profiles $c_{\boldsymbol{\delta}}^1, c_{\boldsymbol{\delta}}^2, \ldots, c_{\boldsymbol{\delta}}^Q$ based on the Euclidean distance criteria. Thus, the total number of labels is P times Q. The combination of the clusters, to which $\boldsymbol{a}_{\mathrm{X},n}^*$ and $\boldsymbol{\delta}_n^*$ belongs, decides the label for a scenario $\{\mathbf{M}, \boldsymbol{\theta}\}_n$. Here, the one-hot vector $\boldsymbol{y}_n$ is constructed of size P times Q and the element which represents the combination of clusters to which the best trajectory belongs is assigned a value 1. Thus, $\{\mathbf{M}, \boldsymbol{\theta}\}_n$ is used as an input and $\boldsymbol{y}_n$ is used as a label for training a 3D-ConvNet in the HARRT+ algorithm.

Evaluation Metric

A *top-i* classification error, which tests if the reference class was within i hypotheses having the highest probability, is used in this work. This metric is suitable as the goal of this work is to find top m classes out of the total L classes. The HARRT algorithm uses only predicted best top m longitudinal acceleration profiles while the ARRT+ algorithm uses only predicted best top m cluster combinations of the steering wheel angle and longitudinal acceleration profiles. The softmax classifier in the 3D-ConvNet architecture gives proba-

bilities for each class, which are used to get top m classes with the highest probabilities.

3D-ConvNet Architecture

Three 3D-ConvNet models with different architectures are designed by changing the number of layers and the number of kernels. These are used to train acceleration profile classifiers for both curved road and intersection road designs. The dimensions of kernels in the respective convolution layer are not changed and all convolutional layers are followed by the ReLU activation function and average pooling in all architectures. The architecture providing best results with the least number of parameters is finally selected.

The results in Table 5.1 shows that networks with two convolution layers and eight filters in each convolution layer, as shown in Fig. 5.5, have the least *top*-2 and *top*-3 errors, and also have the least number of parameters which makes them most efficient in terms of memory. This architecture is further used to measure the efficiency of the 3D-ConvNet method with the top 3 acceleration profiles.

Table 5.1: Comparison of 3D-ConvNet Architectures.

Architecture	**1 Convolution**	**2 Convolution**	**2 Convolution**
Number of Kernels in Convolutional Layer	8	8	12
Number of Learnable Parameters	47934	17916	39524
Curved Road			
-*top*-3 (%) error	4.14	2.21	3.89
-*top*-2 (%) error	18.34	9.32	11.64
Intersection			
-*top*-3 (%) error	17.62	13.72	14.9
-*top*-2 (%) error	30.13	20.82	26.36

In this architecture, shown in Fig. 5.5, 10 predicted occupancy grids $\{\mathcal{G}_{t_0+\Delta t}, \ldots, \mathcal{G}_{t_0+\tau_1}\}$ of the size 30×50 meters, each cell of dimension 1×1 meter, over the prediction time interval $[t_0 + \delta t, t_0 + \tau_1]$ are used as input. τ_1 and δt are chosen to be 2 seconds and 0.2 seconds, respectively. 3D convolutions with 8 kernels of size $4 \times 6 \times 3$ (4×6 in the spatial dimension and 3 in temporal dimension) are applied to generate 8 sets of feature maps $C1$ of size $27 \times 45 \times 8$. The number of trainable parameters in this layer is 584. It is followed by $3 \times 3 \times 2$ pooling on each feature map of $C1$ layer to generate the same number of feature maps with the reduced dimension of $9 \times 15 \times 4$. There are no trainable parameters for this layer. In the second convolutional layer, the feature maps $C2$, with each feature map of dimension $6 \times 12 \times 2$, is obtained by applying convolutions with 8 kernels of size

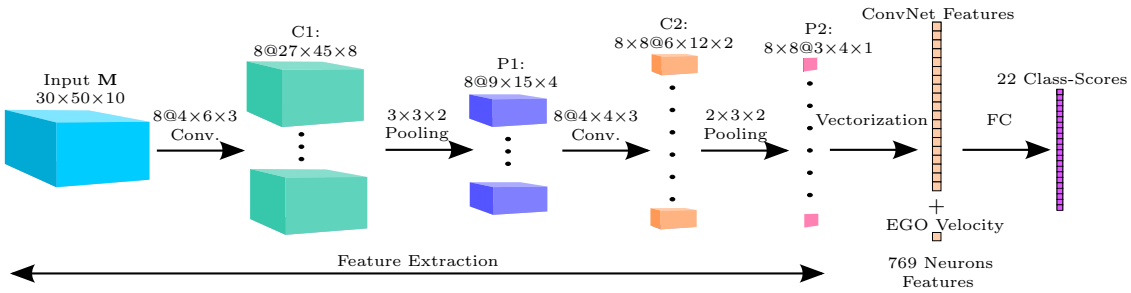


Figure 5.5: 3D-ConvNet architecture used in HARRT Algorithm.

$4 \times 4 \times 3$. The number of trainable parameters for this layer is 392. The number of feature maps generated is 64 as 8 kernel has applied convolution on 8 $P1$ feature maps. A $2 \times 3 \times 2$ pooling is again performed on each of these feature maps to generate 64 feature maps of size $3 \times 4 \times 1$. Finally, all these feature maps are vectorized and the EGO vehicle initial velocity is added as a feature $\boldsymbol{\eta}$ to form a vector of 769 numbers, which is fully-connected to the output layer. Here, the EGO vehicle initial velocity is used as in the simulated traffic-scenarios only this parameter of the EGO vehicle is changed. If other parameters such as acceleration, size is also changed, then those should be added as well. 22 class-scores, which correspond to 21 longitudinal acceleration profiles used with the ARRT algorithm and one for the class "No safe trajectory found", are calculated by matrix multiplication followed by the bias offset and a softmax classifier. The number of trainable parameters in this final part of the classifier is 16940. The total number of learnable parameters in the whole network is 17916. Table 5.2 shows the number of learnable parameters in each layer of the 3D-ConvNet used in the HARRT algorithm.

Table 5.2: Learnable Parameters in 3D-ConvNet.

Layer	**Number of Parameters**
C1	584
P1	0
C2	392
P2	0
FC1	16940
Total	17916

The ConvNet architecture used in the HARRT+ algorithm is the same as the one used in the HARRT algorithm except with an additional fully-connected layer (FC2) and a change in the number of output neurons as shown in the Fig. 5.6. 4 clusters of both the longitudinal acceleration and steering wheel angle profiles are used to form 16 output classes. The number of learnable parameters in each layer of the 3D-ConvNet is shown in Table 5.3. A 3D-ConvNet is trained for predicting the longitudinal acceleration profile cluster and the steering wheel angle profile cluster for curved road scenarios. A data for 44692 critical

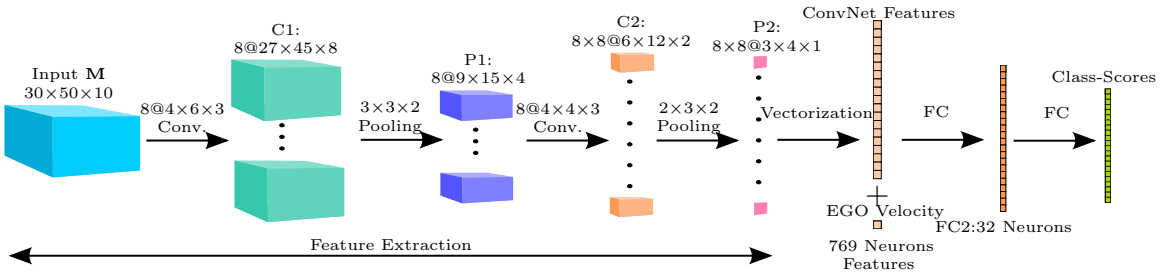


Figure 5.6: 3D-ConvNet Architecture for the HARRT+ Algorithm.

traffic-scenarios is generated out of which 80% data is used as the training data and the remaining 20% as the validation data. The 3D ConvNet achieved the maximum *top-3* accuracy of 83.66% on the validation data for the HARRT+ algorithm.

Table 5.3: Learnable Parameters in 3D-Convet.

Layer	**Number of Parameters**
C1	584
P1	0
C2	392
P2	0
FC1	24640
FC2	528
Total	26144

5.3.2 Hybrid Augmented CL-RRT (HARRT) Algorithm

The procedure for finding and choosing a safe trajectory in the HARRT algorithm is visualized in Fig. 5.7. Initially, when a critical scenario is detected, it is converted into a sequence of predicted occupancy grids $\mathbf{M}$ using suitable models described in **??**ChapDMC) for road traffic-participants. The ARRT algorithm uses all L acceleration profiles, while the HARRT algorithm uses only predicted m acceleration profiles with the ARRT algorithm to find safe trajectories from which it selects the best trajectory.

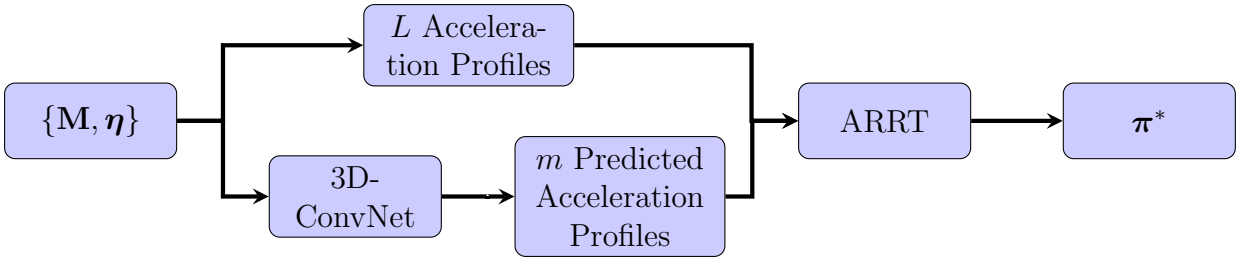


Figure 5.7: Comparison Between ARRT and HARRT Algorithms.

5.3.3 Hybrid Augmented CL-RRT+ (HARRT+) Algorithm

A biased-sampling procedure might increase the convergence speed of an RRT algorithm. However, it risks the primary benefits of randomization in the RRT algorithms. Therefore, a good sampling strategy for search algorithms is the suitable trade-off between randomized and deterministic sampling. The HARRT algorithm does not have enough randomness in the sampling of the longitudinal acceleration as only a few predicted predefined longitudinal acceleration profiles are used. On the other hand, it does not have any bias in the lateral acceleration apart from some bias towards the goal region S_{goal}. The ARRT+ algorithm has randomness in sampling both the longitudinal and the lateral acceleration. The HARRT+ algorithm uses a trained 3D-ConvNet to predict the longitudinal acceleration cluster $\hat{c}^p_{\boldsymbol{a}}$ and the steering wheel angle cluster $\hat{c}^q_{\boldsymbol{\delta}}$. Further, it generates a set of waypoints from these predicted clusters, which it utilizes to bias the sampling in the lateral dynamics of the vehicle and it also biases the sampling in the longitudinal dynamics of the vehicle using the predicted cluster $\hat{c}^p_{\boldsymbol{a}}$. This section describes this biased-sampling strategy in detail.

Generation of Waypoints W

The label y_n for an input $\{\mathbf{M}, \boldsymbol{\eta}\}_n$ represents the combination of the longitudinal acceleration cluster $c^p_{\boldsymbol{a}}$ and the steering wheel angle cluster $c^q_{\boldsymbol{\delta}}$. This means that the acceleration and steering angle profile for the safe trajectory of the scenario $\{\mathbf{M}, \boldsymbol{\theta}\}_n$ lies in $c^p_{\boldsymbol{a}}$ and $c^q_{\boldsymbol{\delta}}$, respectively. These clusters contain many profiles for the time interval $[t_0, t_0 + \tau_1]$. As a representation of these clusters, mean values and standard deviations are calculated [VF04] at every time instant within the time interval $[t_0, t_0 + \tau_1]$ and stored in the mean value vectors $\boldsymbol{\mu}_{c^p_{\boldsymbol{a}}}$, $\boldsymbol{\mu}_{c^q_{\boldsymbol{\delta}}}$ and in the standard deviation vectors $\boldsymbol{\sigma}_{c^p_{\boldsymbol{a}}}$, $\boldsymbol{\sigma}_{c^q_{\boldsymbol{\delta}}}$. Hence, the labels can be represented as

$$y_n \Leftrightarrow [c^p_{\boldsymbol{a}}, c^q_{\boldsymbol{\delta}}] \Leftrightarrow [(\boldsymbol{\mu}_{c^p_{\boldsymbol{a}}}, \boldsymbol{\sigma}_{c^p_{\boldsymbol{a}}}), (\boldsymbol{\mu}_{c^q_{\boldsymbol{\delta}}}, \boldsymbol{\sigma}_{c^q_{\boldsymbol{\delta}}})]. \tag{5.3}$$

Waypoints can be generated as shown in the Fig. 5.8 for a scenario $\{\mathbf{M}, \boldsymbol{\eta}\}$. A trained 3D-Convet predicts the label $\hat{y}$ which can also be represented with $(\hat{\boldsymbol{\mu}}_{c^p_{\boldsymbol{a}}}, \hat{\boldsymbol{\sigma}}_{c^p_{\boldsymbol{a}}})$ and $(\hat{\boldsymbol{\mu}}_{c^q_{\boldsymbol{\delta}}}, \hat{\boldsymbol{\sigma}}_{c^q_{\boldsymbol{\delta}}})$ as per the Eq. 5.3. These are used as the input for a single-track kinematic vehicle model

to generate a trajectory $\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{a}}^p}, \hat{\boldsymbol{\mu}}_{c_{\boldsymbol{\delta}}^q})$. A complex two-track vehicle dynamic model is not necessary as the purpose is to find just waypoints in order to generate a bias for the sampling-based algorithms. The coordinates of this trajectory at the time-steps $t_0 + n\Delta t$, where $n = 1, 2, \ldots, (\tau_1/\Delta t)$, are stored in a matrix called waypoints, $\mathbf{W} \in \mathbb{R}^{2,n}$.

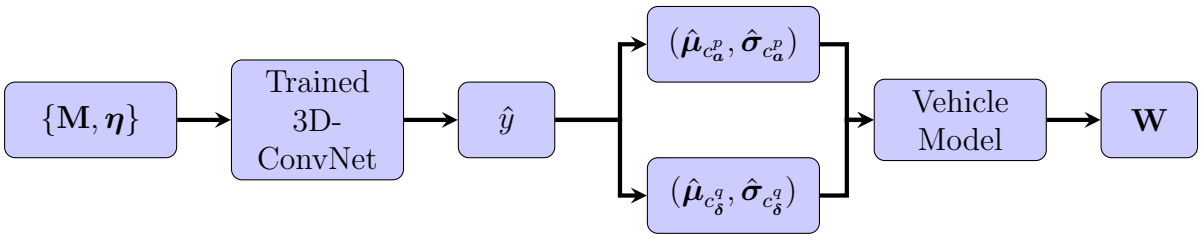


Figure 5.8: Waypoints Geenration.

Biased-Sampling Methods

The sampling of random states s_{rand} for the HARRT+ algorithm can be biased by sampling around waypoints $\mathbf{W}$ as they represent the approximate trajectory predicted by the machine learning algorithm. However, the trajectory coordinates in $\mathbf{W}$ will serve for the biased-sampling only in the lateral dynamics of the vehicle. For the biased-sampling in the longitudinal dynamics, the HARRT+ algorithm samples the longitudinal acceleration $a_{\mathbf{r}_i^k}$ for the state $\mathbf{r}_i^k$ based on the means and variances that are computed by the machine learning algorithm. Instead of sampling uniformly within the possible range of the longitudinal acceleration values like in the ARRT+, it samples $a_{\mathbf{r}_i^k}$ from the Gaussian distribution with mean $\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{a}}^p}(t_{\mathbf{r}_i^k})$ and standard deviation $\hat{\boldsymbol{\sigma}}_{c_{\boldsymbol{a}}^p}(t_{\mathbf{r}_i^k})$ as

$$a_{\mathbf{r}_i^k} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{a}}^p}(t_{\mathbf{r}_i^k}), \hat{\boldsymbol{\sigma}}_{c_{\boldsymbol{a}}^p}(t_{\mathbf{r}_i^k})), \tag{5.4}$$

where, the time $t_{\mathbf{r}_i^k}$ of the new state $\mathbf{r}_i^k$ is obtained from the time parameter $t_{\mathbf{r}_{i-1}^k}$ of the parent state $\mathbf{r}_{i-1}^k$ from which an extension is made as per Eq. 3.14. The HARRT+ algorithm also checks both stable profile and actuator constraints for the sampled acceleration value $a_{\mathbf{r}_i^k}$ as explained in Section 3.6. This simultaneous biased-sampling of states around waypoints $\mathbf{W}$ and the longitudinal acceleration $a_{\mathbf{r}_i^k}$ is termed as *the simultaneous biased-sampling in lateral and longitudinal dynamics*.

Another way of the biased-sampling is based on deterministically sampling center point of goal region S_{goal} instead of the random sample s_{rand} and the random acceleration. This serves as the biased-sampling only in the lateral dynamics of the vehicle and called as *goal-biased sampling*.

Sampling Strategy for the HARRT+ Algorithm

The sampling strategy for the HARRT+ algorithm aims to have an suitable trade-off between randomized and deterministic sampling. There are three phases in this sampling

procedure shown in Fig. 5.10. In the first phase, it only uses the *simultaneous biased-sampling in lateral and longitudinal dynamics* to find the initial states of the tree. It is because the subsequent growth of the tree largely depends on the initial growth of the tree. The second phase comprises of the sequential use of the *simultaneous biased-sampling in lateral and longitudinal dynamics*, the *goal-biased sampling*, and the uniform sampling. In the final phase, this algorithm switches to the uniform sampling and every third sample to the *goal-biased sampling* only for every third sample.

The benefit of this sampling strategy can be understood from Fig. 5.9 and 5.11 that show an example of the trajectory planning using the HARRT+ algorithm with top 2 predictions of the 3D-ConvNet in the same traffic-scenario. It is a scenario with a two-lane road where a collision of the EGO vehicle is predicted with a pedestrian crossing the road. The EGO vehicle is moving witha velocity of 36 km/hr while all other vehicles are moving with a velocity of 50 km/hr. The trained 3D-Convet predicts the best class which is represented by $\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{a}}^p}$ and $\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{\delta}}^p}$, shown in Fig. 5.9b and 5.9c in green color, respectively. They are used to generate a trajectory $\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{a}}^p}, \hat{\boldsymbol{\mu}}_{c_{\boldsymbol{\delta}}^q})$ from which waypoints $\mathbf{W}$ are obtained. The HARRT+ algorithm is used to get $\boldsymbol{\pi}^*$. The acceleration $\boldsymbol{a}_{\boldsymbol{x}}^*$ and the steering angle $\boldsymbol{\delta}^*$ that are finallyy found by the HARRT+ algorithm are shown in magenta color in Fig. 5.9b and 5.9c. The

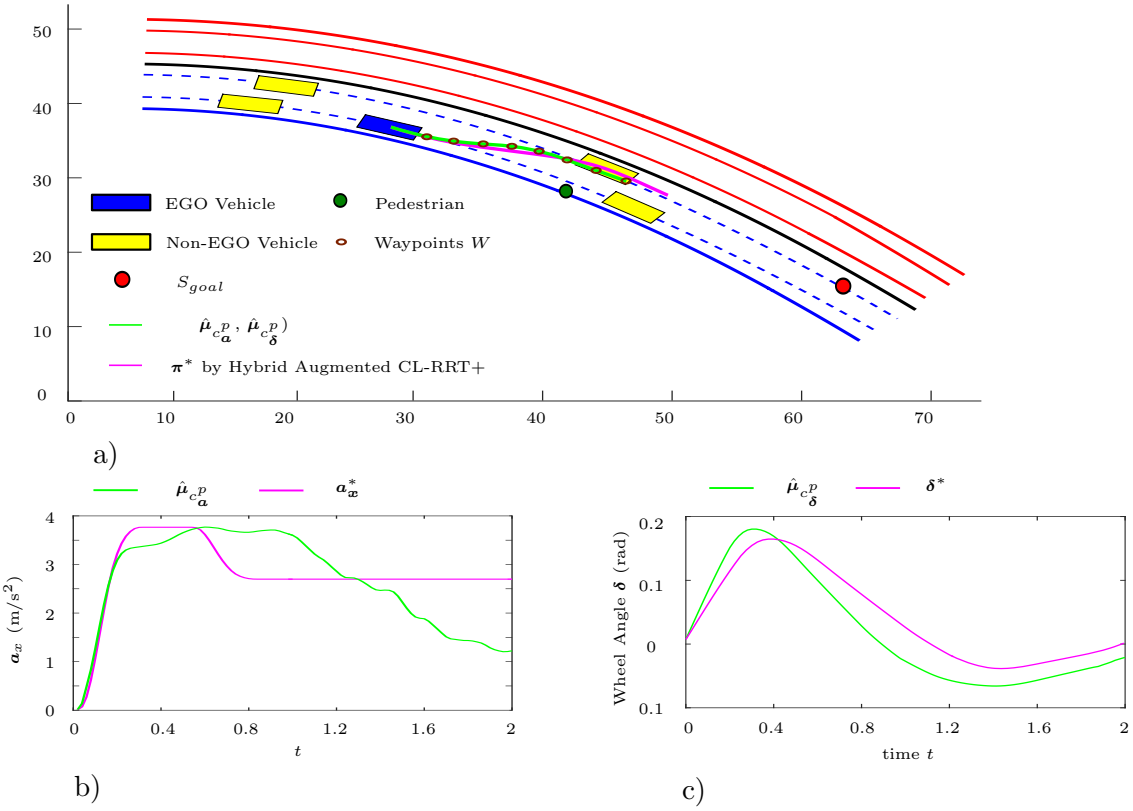
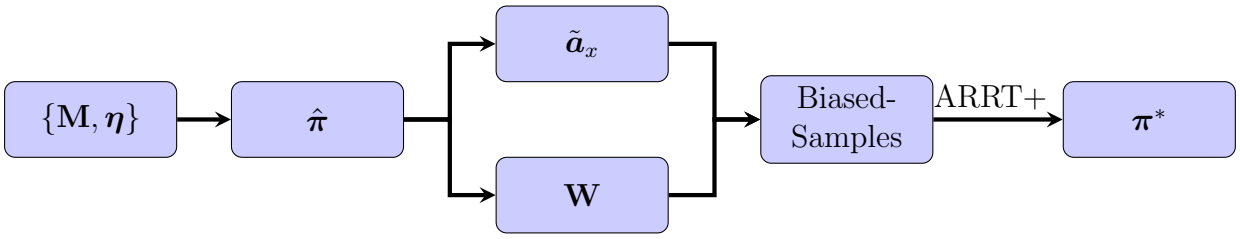
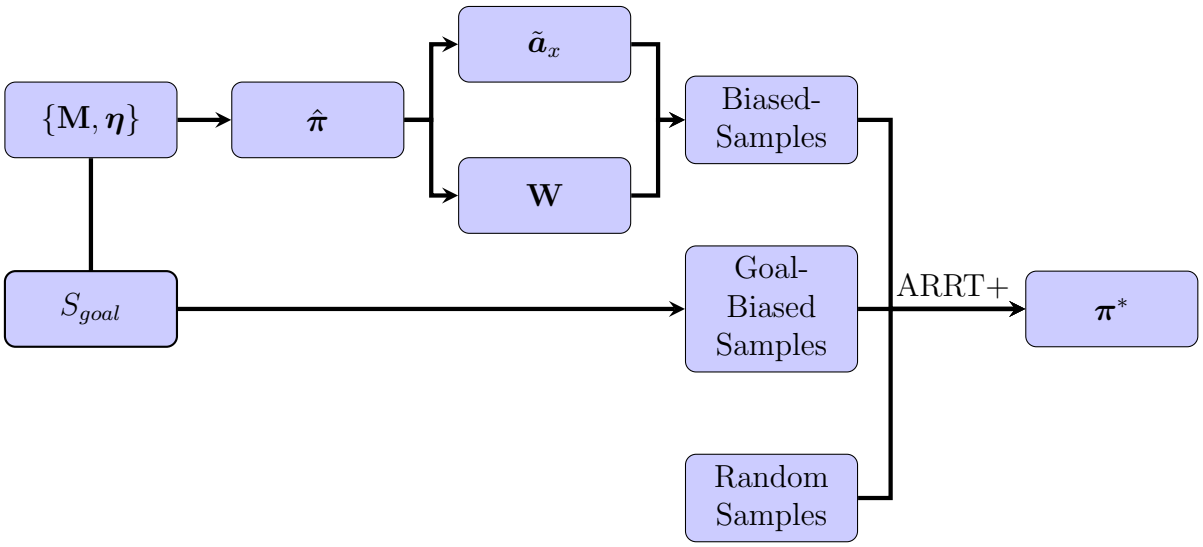


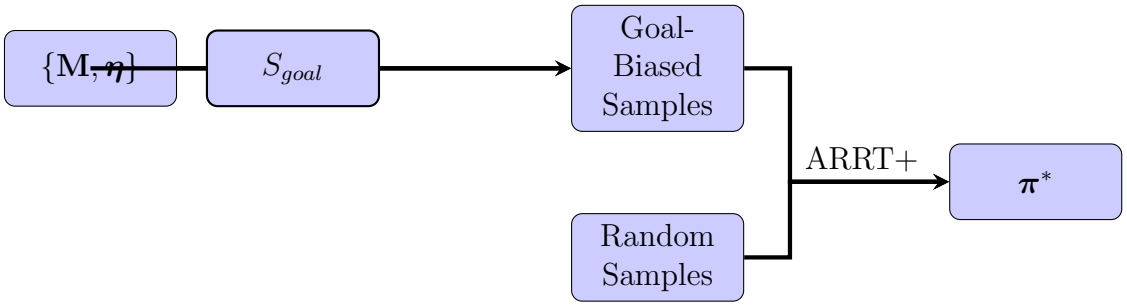
Figure 5.9: Trajectory Planning with the HARRT+ Algorithm using the Best Prediction.



a) Simultaneous Biased-Sampling in Lateral and Longitudinal Dynamics.



b) Simultaneous Biased-Sampling in Lateral and Longitudinal Dynamics, the Goal-Biased Sampling, and the Randomized Sampling.



c) Randomized Sampling with the Goal-Biased Sampling.

Figure 5.10: HARRT+ Algorithm Sampling Strategy.

plots show the biased-sampling based on the $\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{a}}^p}$, $\hat{\boldsymbol{\sigma}}_{c_{\boldsymbol{a}}^p}$ and $\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{\delta}}^p}$ helped in finding a collision-free trajectory even with strong positive acceleration.

In Fig. 5.11, the second-best prediction from the 3D-ConvNet is used to find a safe trajectory. As it can be seen in Fig. 5.11b, the prediction of $\hat{\boldsymbol{\mu}}_{c_{\boldsymbol{a}}^p}$ is wrong. Even then, the HARRT+ algorithm converged because of its sampling strategy, which uses the combination of biased and random sampling, although it needed more samples compared to the previous case with the best prediction.

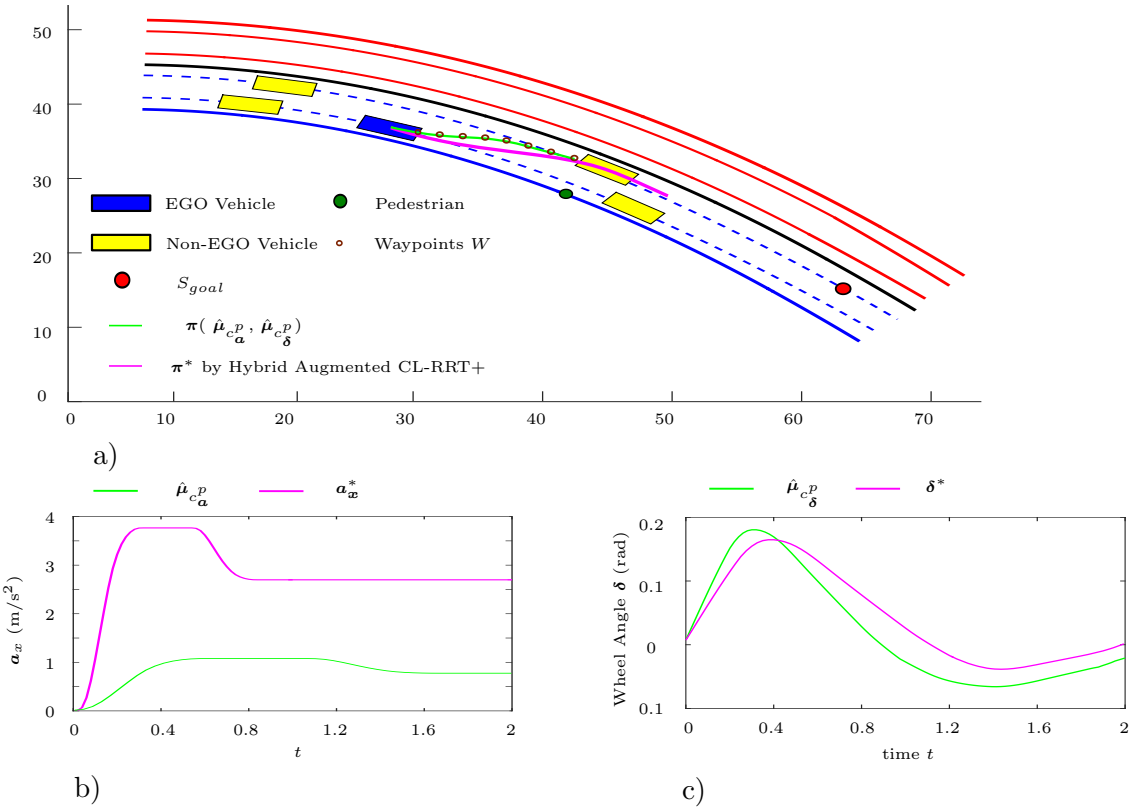


Figure 5.11: Trajectory Planning with the HARRT+ Algorithm using 2nd Best Prediction.

5.3.4 Simulation Results

The proposed hybrid machine learning methods are compared with their corresponding sampling-based methods using two criteria: *efficiency* and *safety*. The efficiency is measured based on the time and number of random samples required for the execution, and the safety is measured based on the success rate, i. e., percentage of scenarios in which the algorithm can find a safe trajectory.

Comparison Between the ARRT and HARRT Algorithm

In total, 1890 random test scenarios for curved roads and 820 random test scenarios for intersection are simulated in the Matlab simulation environment, and the ARRT and HARRT algorithms are used to find safe trajectories. A maximum of 100 samples is used with every acceleration profile to find a safe trajectory.

Table 5.4: Comparison Between The ARRT and HARRT Algorithm.

	Curved Road		**Intersection**	
Criteria	**ARRT**	**HARRT**	**ARRT**	**HARRT**
Average Samples	1272	109	1868	237
Average Time (Sec.)	5.0518	0.6230	6.6989	1.3126
Collision-free Trajectory Found (%)	100	99.63	60.85	51.95
Nonsevere Trajectory Found (%)	0	0.37	39.15	42.37

Table 5.4 shows the results of the simulation with both methods. The average number of samples required by the ARRT algorithm in the curved road and intersection scenarios is 1272 and 1868, respectively. On the other hand, the HARRT+ algorithm using the 3D-ConvNet needed only 109 and 237 samples in both road designs on an average. Thus, the hybrid method shows 11.67 and 7.88 times improvement in terms of the number of samples required for the safe trajectory planning in the curved road and intersection scenarios, respectively. Similarly, a comparison for the time required by both methods in the curved road and intersection scenario shows 8.11 and 5.1 times improvement, respectively. The time required for constructing a sequence of predicted occupancy grids $\mathbf{M}$ is included in the time required for finding safe trajectories by the hybrid method in order to assure a fair comparison. As only 3 instead of 21 acceleration profiles are used, a reduction of 7 times in the number of samples and the time is expected. However, suitable acceleration profiles require much less than 100 samples for finding a collision-free trajectory. Therefore, the proposed approach shows an improvement of more than 7 times. Only intersection scenarios show less than 7 times improvement in terms of time required as many samples lead to collision states, which further requires the computation of the severity of injury as described in the Section 3.5.6.

Table 5.4 also shows the percentage of scenarios in which both algorithms are able to find safe trajectories. For the curved road design, the brute force analytical method was able to find 100% collision-free trajectories. Although the hybrid method is able to find

a collision-free trajectory in 99.63% scenarios, in the rest of 0.37% scenarios, it can find a nonsevere trajectory. Similarly, in intersection scenarios, the brute force analytical method was able to find a safe trajectory in 100% scenarios again, but HARRT was able to find a safe trajectory in 94.32% scenarios.

Comparison Between the ARRT+ and HARRT+ Algorithm

A number of test traffic-scenarios were simulated with the ARRT+ algorithm and the HARRT+ algorithm. These test traffic-scenarios are divided into two categories, *training curves test data* and *non-training curves test data*. The *training curves test data* are the traffic-scenarios which are different than those in the training and the validation data, but on the same curves. The *non-training curves test data* are the traffic-scenarios on different curves than those used for generating the training and the validation data. 2100 random samples were used for finding a safe trajectory with the ARRT+ algorithm. In the HARRT+ algorithm, top 3 labels were predicted by a trained 3D-ConvNet and 100 samples were used with each label to find a safe trajectory out of which the first 10 were used in the first phase, the next 60 in the second phase, and the remaining 30 for the final phase of the sampling strategy.

Table 5.5: Comparison Between The ARRT+ and HARRT+ Algorithm.

	Training Curves Test data (994 Scenarios)		**Non-Training Curves Test data** (403 Scenarios)	
Criteria	**ARRT+**	**HARRT+**	**ARRT+**	**HARRT+**
Average # States	163	101	186	112
Average Time (sec.)	4.06	1.03	4.26	1.27
Collision-free Trajectory Found (%)	96.50	95.83	96.66	90.60
No Safe Trajectory Found (%)	0	0.80	0.99	1.74

Table 5.5 shows the results of the simulation. The HARRT+ algorithm is more efficient than the ARRT+ algorithm in terms of both the memory and the computation time and was equally good as the ARRT+ algorithm in finding safe trajectories for *training curves test data*. Although curved roads used for generating *non-training curves test data* are not used for generating any training data, the HARRT+ still gives very good results. The time required for constructing a sequence of predicted occupancy grids $\mathbf{M}$ is included in the

time required for finding safe trajectories by the HARRT+ algorithm in order to assure a fair comparison.

5.3.5 Drawbacks of the HARRT and HARRT+ Algorithm

The hybrid machine learning algorithms HARRT and HARRT+ use machine learning algorithms to assist their corresponding sampling-based trajectory planning algorithms ARRT and ARRT+, respectively. The role of a machine learning algorithm in the HARRT algorithm is to reduce the possible search space for the ARRT algorithm by predicting the best acceleration profiles and only using them. It affects the probabilistic completeness of the ARRT algorithm in case a machine learning algorithm predicts wrong classes, i. e., wrong longitudinal acceleration profiles. There are traffic-scenarios in which with only a fixed longitudinal acceleration profile (e. g. strong breaking profile) a safe trajectory can be found, but if the machine learning algorithm does not predict this profile, a safe trajectory will not be found by the HARRT algorithm even though a solution exists.

This drawback of the HARRT algorithm, lacking probabilistic completeness, is solved in the HARRT+ algorithm. The role of a machine learning algorithm in the HARRT+ algorithm is to find regions in search space where the probability of finding the final solution is high. Then the HARRT+ algorithm samples in these regions with high density. Thus, the HARRT+ algorithm uses a combination of biased and random sampling algorithm. Therefore, it still has the property of probabilistic completeness even with wrong predictions of template trajectory $\hat{\boldsymbol{\pi}}_t$. However, the computation time for finding a safe trajectory will be high when it predicts a wrong cluster combination because of the wrong bias generation. Even if a right template trajectory is predicted and the final safe trajectory lies on the boundary of these clusters, it is still difficult for the HARRT+ algorithm to converge. In such situations as well, it is observed that the HARRT+ algorithm converges slowly.

Another critical factor for the convergence of RRT algorithms, which is not extensively researched, is the sequence of sampling. There is an inherent bias in the extension of trees in regions near the states already presented in the tree, which makes it essential how the tree is grown in initial iterations. A single wrong extension of the tree in the early stage can have a significant influence on the success of convergence of the algorithm. The HARRT+ algorithm uses biased samples initially as it assumes that the predicted trajectory by the machine learning algorithm is close to the final solution. With false prediction, there will be a wrong bias generated, which in turn puts the trajectory planning algorithm at risk. Also, finding early states of the tree using biased samples at the tale of the predicted trajectory is equivalent to the extension of trees using samples generated from wrong bias.

The performance of the HARRT+ algorithm mainly depends on the number of clusters of the longitudinal acceleration and steering wheel angle profiles. With few clusters, the size of the clusters grows, which in turn reduces the effect of biased sampling as the safe trajectory could lie anywhere in the predicted cluster, sometimes at the boundary of the cluster as well. The biased-sampling would be much more effective with smaller clusters

provided they are correctly predicted. However, with smaller clusters, the number of clusters increases, which in turn, also increases the number of output labels for machine learning. This leads to a lower accuracy of the machine learning algorithm. Otherwise, a more complex machine learning model needs to be designed to attain the same level of accuracy, which means an increase in the required computational resources.

The original problems associated with randomized sampling-based algorithms, such as the uncertainty in finding a solution and the lack of repeatability, exist with both the HARRT and HARRT+ algorithm as both algorithms still have randomness in their nature. Also, these algorithms give priority to find feasible trajectory planning and perform optimal trajectory planning only if computational resources are available given a feasible trajectory is already found[1]. Feasible trajectories are the collision-free trajectories, and the optimal trajectories are the collision-free trajectories with high comfort (e. g. low acceleration values) criteria. It is especially problematic when the traffic-scenarios are simple where deterministic algorithms can find an optimal trajectory quickly and sampling-based algorithms either do not converge or find a very complex safe trajectory with harsh interventions even when they are not necessary. Therefore, there is a need to investigate if random sampling-based trajectory planning algorithms can be replaced by a deterministic planning algorithm that can find the safe trajectories that are not just feasible but also optimal.

5.4 Combination of Machine Learning with Deterministic Algorithms

All the variants of the RRT algorithm described in previous sections of this chapter make use of the random sampling strategy with a combination of some deterministic sampling to find safe trajectories in critical traffic-scenarios. They achieve this by sampling and incrementing the trajectory iteratively, i. e., by finding small parts of trajectories in each iteration. In a multidimensional continuous state-space and action-space, this is a hard problem to solve. Therefore machine learning algorithms are designed to assist the algorithms to sample in promising regions. Still, as the search process is iterative, a wrong step in one iteration may affect the subsequent iteration adversely. Another approach is to sample the whole trajectory and modify it gradually in iterations to find the final solution. This is the motivation for the algorithms proposed in this section.

From the explanation of the drawbacks of the HARRT+ algorithm, it is clear that the final computation time required for trajectory planning with the HARRT+ algorithm strongly depends on the quality of the predicted reference trajectory, i. e., the closer the predicted reference trajectory $\hat{\boldsymbol{\pi}}_t$ in distance and shape to $\boldsymbol{\pi}^*$ lesser will be the computation time

[1]Depending on whether the quality of the solution path is considered, the terms feasible and optimal are used to describe this path. [PCY+16] Feasible trajectory planning refers to the problem of determining a trajectory that satisfies some given problem constraints without focusing on the quality of the solution. In contrast, optimal trajectory planning refers to the problem of finding a trajectory that optimizes some quality criterion subject to given constraints.

required to find the final trajectory $\boldsymbol{\pi}^*$. It is not possible to find a good quality reference trajectory in all scenarios with the finite number of template trajectories. Ideally, the machine learning algorithm should be capable of predicting the trajectory as close as possible to the final solution, which means there should be many, ideally infinite, template trajectories. This will change the output of machine learning algorithm from a finite number of labels to a real number, i. e., the machine learning task would change from classification to regression.

The above requirements motivate the need for an algorithm that can sample trajectories and modify those trajectories iteratively to find a safe trajectory. Therefore, this section proposes a generative model for trajectory generation using *Variational Autoencoder* (VAE) that can generate many reference trajectories. The different dimensions of the latent vector $\boldsymbol{z}$ in VAE encode interpretable factors of variations in trajectories. Therefore, the generated trajectories can be modified by changing the values of these latent variables. This trained VAE on trajectories also generates targets for the machine learning algorithm, i. e., 3D-ConvNet regressor, which maps the traffic-scenarios to the latent vector $\boldsymbol{z}$ of the VAE.

5.4.1 Generative Model for Trajectories $\boldsymbol{\pi}$

In order to train the VAE for trajectories, 60000 different trajectories for time τ_1 (=2 seconds) are generated as the training data using the nonlinear two-track vehicle dynamic model explained in Section 2.1.3. The plot of these trajectories can be seen in the Fig. 5.12. The initial velocities were changed in the range 20-50 km/hr. The lateral and longitudinal dynamic intervention over each trajectory is randomly sampled using actuator and stable actuator profile constraints similarly as described in Section 3.6. These trajectories are provided as input to the VAE in the form

$$\boldsymbol{\pi} = \{r_{x_{t_0}}, r_{x_{t_0+\Delta t}} \ldots, r_{x_{\tau_1}}, r_{y_{t_0}}, r_{y_{t_0+\Delta t}}, \ldots, r_{y_{\tau_1}}\} \ , \tag{5.5}$$

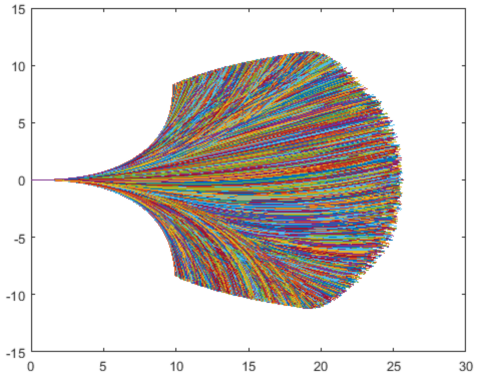


Figure 5.12: Randomly Sampled Trajectories for Training the VAE.

where $r_{x_{t_i}}$ and $r_{y_{t_i}}$ are the coordinates of the center of gravity of the vehicle at time t_i. Fig. 5.13 shows the architecture of the VAE trained on these trajectories. The encoder $q_{\mathbf{z}|\mathbf{x}}(\boldsymbol{z}|\boldsymbol{\pi})$ maps trajectories to the latent space mean vector $\boldsymbol{z}_\mu$ and the standard deviation vector $\boldsymbol{z}_\sigma$ each having a dimension of 2. As per the reparameterization trick, the samples $\boldsymbol{z}$ are obtained by sampling $\boldsymbol{\epsilon}$ from $\mathcal{N}(0,1)$ and performing the operation $\boldsymbol{z}_\mu + \boldsymbol{\epsilon}\boldsymbol{z}_\sigma$. The decoder $f_{\boldsymbol{\pi}|\mathbf{z}}(\boldsymbol{\pi}|\boldsymbol{z})$ reconstructs the trajectories using samples generated from $\boldsymbol{z}_\mu$ and $\boldsymbol{z}_\sigma$. The root mean square criteria is used for the reconstruction loss. Also, the trajectories are normalized before the first layer in the encoder and the output of the decoder is denormalized and a smoothing with moving average filter is performed to get final trajectories $\bar{\boldsymbol{\pi}}$. All the layers in encoder and decoder of the VAE use the hyperbolic tangent as an activation function.

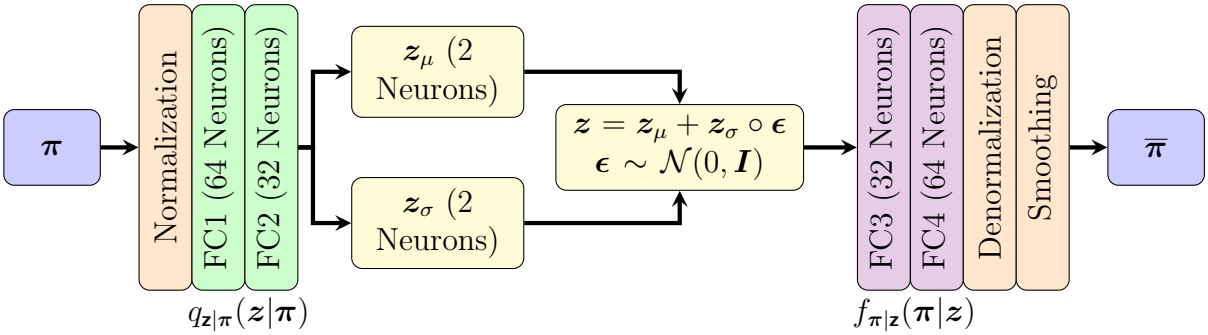


Figure 5.13: VAE Architecture for Trajectories.

5.4.2 3D-ConvNet Regressor

VAE represents the trajectory as a function parameterized by a finite-dimensional latent vector $\boldsymbol{z}$ and a suitable path is sought by optimizing over this parameter vector using nonlinear continuous optimization techniques. This method will converge rapidly to locally optimal solutions. However, it will not find globally optimal solutions unless an appropriate initial guess is provided.

The task of the 3D-ConvNet is to predict the value of the continuous variable $\boldsymbol{z}_\mu$ instead of predicting only finite class labels as in the HARRT+ algorithm. The 3D-ConvNet uses $\{\mathbf{M}, \boldsymbol{\eta}\}$ as the input. The architecture of the 3D-ConvNet is the same as the one used for the HARRT+ algorithm described in Fig. 5.6, except the output size changed to two. The loss function calculation criterion is also changed from the cross-entropy to the root mean square error.

The label generation procedure for the 3D-ConvNet regressor is explained in Fig. 5.14. For each traffic-scenario $\mathbf{M}, \boldsymbol{\eta}\}$, the best trajectory $\boldsymbol{\pi}^*$ is found with the ARRT+ algorithm in the Matlab simulation environment described in Section 2.4. This trajectory is fed to the encoder $q_{\mathbf{z}|\mathbf{x}}$ of trained VAE to find a corresponding $\boldsymbol{z}_\mu$ which is assigned as a target for that scenario. In total 44692 curved road critical traffic-scenarios with different radius of

curvatures, number and type of objects are simulated and corresponding targets are found.

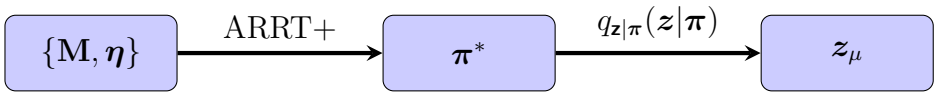


Figure 5.14: Label Generation using VAE.

The inference procedure for 3D-ConvNet is shown in Fig. 5.15. When a traffic-scenario $\{\mathbf{M}, \boldsymbol{\eta}\}$ is encountered, the trained 3D-ConvNet predicts $\hat{\boldsymbol{z}}_\mu$. It is directly fed to the decoder network $f_{\boldsymbol{\pi}|\mathbf{z}}(\boldsymbol{\pi}|\boldsymbol{z})$ of the trained VAE to get the predicted reference trajectory $\hat{\boldsymbol{\pi}}$.

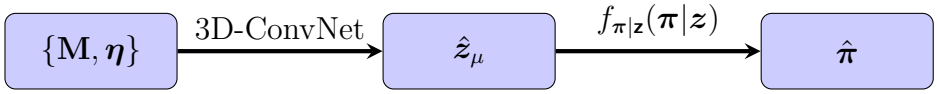


Figure 5.15: Inference using VAE.

5.4.3 Generative Algorithm for Trajectory Exploration (GATE)

The trained VAE can generate trajectories by sampling the latent space values and feeding them to the decoder. Because of the probabilistic nature of VAE, its latent space is continuous unlike in simple autoencoders that use deterministic mapping. This property of VAE can be used for setting up an optimization procedure to find the optimal latent variable values $\boldsymbol{z}^*$, which generates the best trajectory $\boldsymbol{\pi}^*$ using the decoder $f_{\boldsymbol{\pi}|\mathbf{z}}(\boldsymbol{\pi}|\boldsymbol{z})$, from the randomly initialized $\boldsymbol{z}$. The cost function J is defined as per the application based on criterias such as safety, comfort, etc. As the goal of this work is to find trajectories for collision avoidance, the area occupied by the EGO vehicle during the whole trajectory should not intersect with the non-free area, i. e., the area occupied by other road traffic-participants and the area outside of the road. Simultaneously, the criteria of keeping the distance of the EGO vehicle to the othe vehicles as large as possible distance is added so that a small variation in other road participants prediction does not lead to a collision. Therefore, the optimal $\boldsymbol{z}^*$ is found such that

$$\begin{aligned} \boldsymbol{z}^* &= \underset{z}{\operatorname{argmin}} \left[J\right] \\ &= \underset{z}{\operatorname{argmin}} \left[\sum_t (S_{nf}(t) \cap \boldsymbol{s}_{\pi\sim P_{\boldsymbol{\theta}}(\boldsymbol{\pi}|\boldsymbol{z})}(t)) - d_{min)}\right] \quad , \end{aligned} \tag{5.6}$$

where $S_{nf}(t)$ is the non-free area of the road at time t, i. e., the area outside of the road and the area within the road occupied by other road participants at time t, $\boldsymbol{s}_{\pi\sim P_{\boldsymbol{\theta}}(\boldsymbol{\pi}|\boldsymbol{z})}(t)$ is the area occupied by the EGO vehicle at time t along the trajectory $\boldsymbol{\pi}$ obtained by feeding $\boldsymbol{z}$ to the decoder $f_{\boldsymbol{\pi}|\mathbf{z}}(\boldsymbol{\pi}|\boldsymbol{z})$ and d_{min} is the shortest distance between the $\boldsymbol{s}_{\pi\sim P_{\boldsymbol{\theta}}(\boldsymbol{\pi}|\boldsymbol{z})}(t)$ and $S_{nf}(t)$ over the whole trajectory $\boldsymbol{\pi}$ in the time interval $t = [t_0, t_0 + \tau_1]$. The first term on

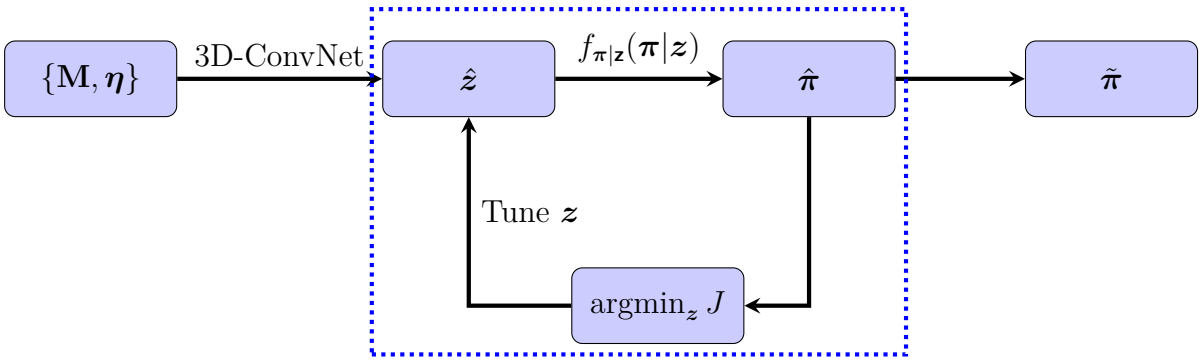


Figure 5.16: GATE Algorithm.

the right hand side of Eq. 5.6, is the summation of the intersection of the non-free area of the road with the EGO vehicle along the trajectory $\boldsymbol{\pi}$. The goal is to make this term zero and increase d_{min}. The optimization solver used is a Matlab imlementation of the Nelder-Mead Simplex method [LRWW98].

The final trajectory obtained by this procedure is highly dependent on the initialization of the latent variable values. With wrong initialization, it may get trapped in a local minima leading to suboptimal values, which could generate a trajectory with a severe collision. Therefore, the trained 3D-ConvNet has to predict the initial values of the latent variables $\hat{\boldsymbol{z}}$, to bes already very close to the optimal value$\boldsymbol{z}^*$. This whole procedure is shown in Fig. 5.16 and this algorithm is named as *Generative Algorithm for Trajectory Exploration* (GATE).

5.4.4 GATE-ARRT+

Although GATE provides an opportunity to sample trajectories directly, it is not a probabilistically complete algorithm like the RRT algorithm. It is because RRT has randomness in its nature, while GATE is completely deterministic except the random initialization of $\boldsymbol{z}$. It can generate random trajectories by sampling continuously latent values $\boldsymbol{z}$. However, this is not sufficient because VAE only learns the approximate training data distribution and not the true data distribution. The generation capacity of the VAE depends on the provided training data and the capacity of the model. However, the reference trajectory generated by GATE can be used to bias the sampling of the ARRT+ algorithm to increase its convergence rate. This combination is named as the GATE-ARRT+ algorithm. It extracts the waypoints and the acceleration profile from the reference trajectory and generates biased-samples for both lateral and longitudinal dynamic intervention. It also uses the same sampling strategy as in the HARRT+ algorithm described in Section 5.3. As the reference trajectories generated by GATE are in most cases closer to the best trajectories compared to the reference trajectories predicted in the HARRT+ algorithm, the GATE-ARRT+ algorithm converges usually more rapidly.

5.4.5 Simulation Results

In order to validate the effectiveness of the proposed vehicle motion planning algorithms, many different curved-road traffic-scenarios with the different number of objects having different initial velocities and positions are simulated in the Matlab simulation environment and safe trajectories with different motion planning algorithms such as ARRT+, HARRT+, GATE and GATE-ARRT+ are found. The search for a collision-free trajectory is stopped when a collision-free trajectory is found or when a maximum number of samples is reached. The maximum number of samples used for the ARRT+ algorithm is set to 2100, as it uses simple random sampling while for the HARRT+ and GATE-HARRT+ 300 samples are used. The number of iterations for the optimization procedure is limited to 10 and 2 with the GATE and GATE-ARRT+ algorithm, respectively. Although the GATE-ARRT+ algorithm does not necessarily require optimization iterations, they improve the predicted trajectory and ease the task of the ARRT+ algorithm. The quantitative results are summarized in Table 5.6. The results show that in scenarios with the fewer objects, the GATE algorithm is able to find a collision-free trajectory in almost all traffic-scenarios with the shortest computation time because lots of free space is available. As the number of objects increases, the free space available decreases, and therefore the GATE algorithm converges in a lesser number of traffic-scenarios. In such cases, the GATE-ARRT+ algorithm is proven to be more effective. The higher efficiency of the GATE-ARRT+ algorithm compared to the HARRT+ algorithm is because of the better reference trajectory provided by the 3D-ConvNet regressor and VAE.

Fig. 5.17a, 5.17b and 5.17c show the safe trajectories planned with GATE-ARRT+, HARRT+ and GATE in a traffic-scenario where a collision with a pedestrian who is crossing the street is predicted. From these figures, it is clear that the HARRT+ algorithm required more samples compared to the GATE-ARRT+ algorithm. Also, the final tra-

Table 5.6: Comparison of Vehicle Motion Planning Algorithms.

		ARRT+	**HARRT+**	**GATE**	**GATE-ARRT+**
# Samples		2100	300	-	300
# Iterations		-	-	10	2
1-2 Objects (834 scenarios)	**Time (Sec.)**	3.33	0.81	0.31	0.45
	% Conv.	97.52	96.04	98.92	97.24
3-4 Objects (1728 scenarios)	**Time (Sec.)**	3.62	1.08	0.83	0.68
	% Conv.	92.99	91.14	72.22	93.17
5-6 Objects (3625 scenarios)	**Time (Sec.)**	4.34	1.32	1.15	0.87
	% Conv.	89.02	86	57.98	88.02

jectory (longest black trajectory) found by the GATE-ARRT+ algorithm has a smoother shape compared to the ones found by HARRT+ and GATE algorithm. This example shows that a better reference trajectory indeed will lead to a final trajectory with a better quality, that will be found by the ARRT+ algorithm.

5.4.6 Comparison of Hybrid Machine Learning Algorithms with Reinforcement Learning

Reinforcement learning, also an area of machine learning, is about taking suitable actions to maximize the reward in a particular robot state $\boldsymbol{s}_t$. It is employed to find the best possible behaviour or path it should take in that robot state. Reinforcement learning differs from the supervised learning in a way that the model is trained with the correct output label itself, whereas in reinforcement learning, there is no output label, but the agent decides

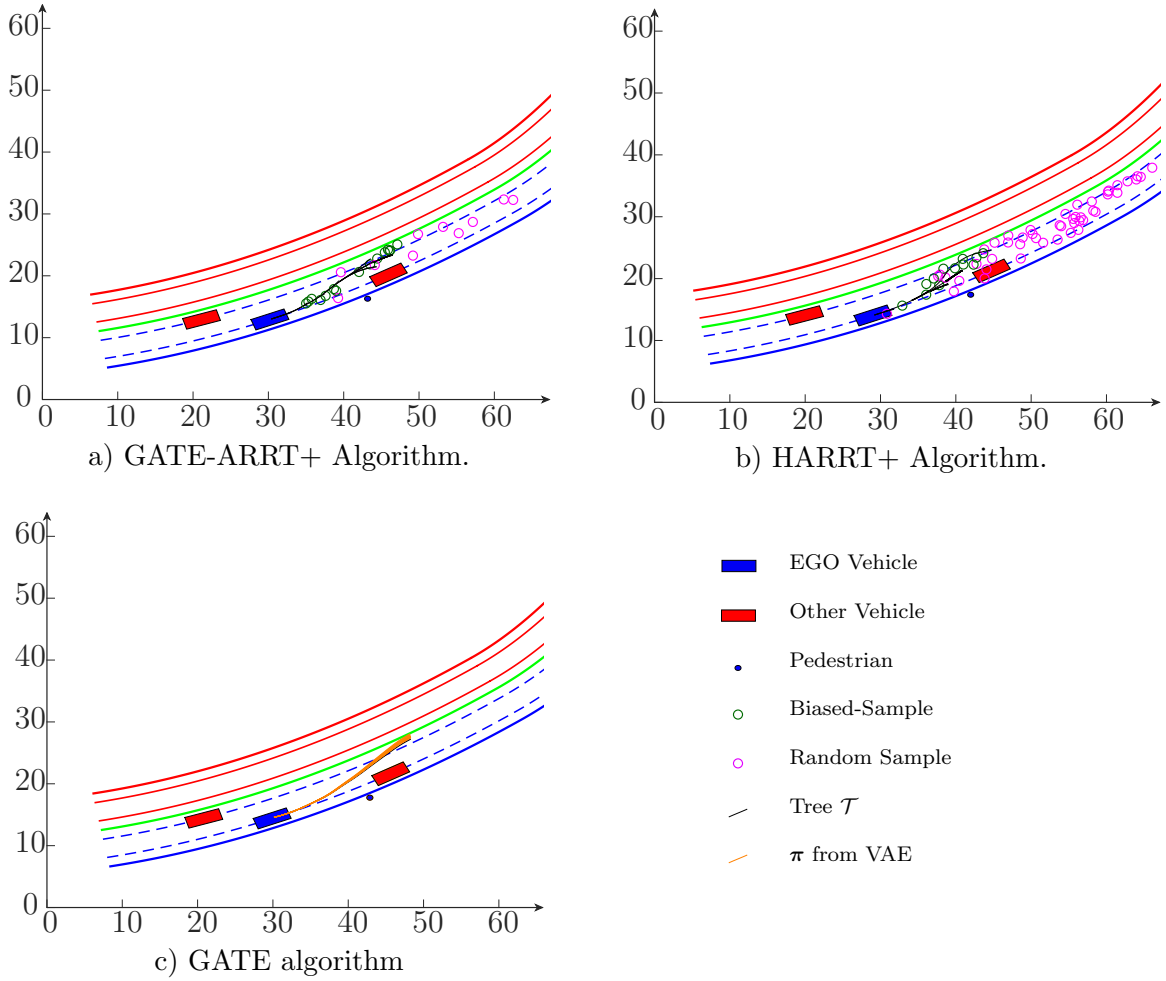


Figure 5.17: Trajectory Planning with the GATE-ARRT+, HARRT+ and GATE Algorithms.

what to do for the given task. In the absence of a training dataset, it learns from its experience.

The trajectory planning algorithms proposed in this chapter use supervised and unsupervised machine learning algorithms, but the methodology of hybrid machine learning has also some similarities to reinforcement learning algorithms. Nevertheless, it differs fundamentally from reinforcement learning algorithms so that it cannot be categorised as reinforcement learning. This section explains these similarities and dissimilarities.

The striking common feature between hybrid machine learning algorithms (except GATE) and reinforcement learning algorithms is that they use the paradigm of "exploration and exploitation". The exploration focuses on new possibilities, whereas exploitation focuses on old certainties. In general, exploitation is associated with determinism, while exploration is the randomization. Many reinforcement learning algorithms use ϵ-greedy strategy to find the optimal trade-off between exploiting and exploring while the hybrid machine learning algorithms proposed in this work use different sampling strategies based on a combination of biased and random sampling. The principal of exploring is in both cases a random choice of an action or a sample. However, there is a fundamental difference in the nature of exploitation in these two methodologies. The exploitation in reinforcement learning is the selection of the best possible action in a given state. For the introduced hybrid machine learning algorithms consist of either sampling close to the predicted solution, as in the HARRT+ and GATE-ARRT+ algorithm, or picking only a subset of actions like in the HARRT algorithm.

Most of the literature about the exploration and exploitation trade-off in reinforcement learning algorithm deals with finding optimal solution. However, the trade-off varies highly depending on the application. Specifically, for the safe trajectory planning in critical traffic-scenarios with multiple static and dynamic objects, where each traffic-scenario is different from one another and the total number of traffic-scenarios is unknown, it is impossible to find the optimal trade-off that will suit all types of traffic-scenarios. Therefore, the focus of this work is to use exploration and exploitation differently. The exploitation is the use of off-line learning from a limited number of traffic-scenarios so that the trajectory planning algorithms converges rapidly. At the same time, exploration is not about gaining new knowledge but just exploring new possibilities when the learned rules failed to converge, e. g., when the traffic-scenario encountered is different from those used when defining off-line learning rules. Therefore, instead of finding an optimal trade-off between exploration and exploitation, the proposed hybrid machine learning algorithms define a switch from exploitation to exploration, i. e., from biased sampling to randomized sampling after a specified number of iterations/samples if a safe trajectory is not found.

Summary

In the first part of this chapter, a literature survey about machine learning applications for trajectory planning algorithms is presented. Further, the details of the hybrid machine learning algorithms, namely HARRT, HARRT+, GATE, GATE-ARRT+, along with sim-

ulation results, are described. The simulation results show that hybrid machine learning algorithms are more efficient than their corresponding analytical trajectory planning algorithms without compromising safety. Specifically, the GATE-ARRT+ has shown the best results in all types of critical traffic-scenarios. Finally, the proposed methodology of hybrid machine learning algorithms is compared with reinforcement learning algorithms to specify how the former is different from later.

Chapter 6

Optimization Methods for Trajectory Planning Algorithms

The high accuracy of deep neural network algorithms comes at the cost of high computational complexity. Therefore, computational engines such as GPU, ASIC, FPGA are used for applications with deep neural network implementations. Therefore, even if the hybrid machine learning algorithms, such as the HARRT, HARRT+, GATE and GATE-ARRT+ algorithms explained in Chapter 5, reduce the computation time, the total time and memory requirements are still high for the implementation on an automotive microcontroller. An efficient embedded implementation of these algorithms is required as the vehicle onboard microcontroller resources are limited. This chapter proposes machine learning and analytical approaches for replacing the computationally intensive modules common to all trajectory planning algorithms proposed in this work. It also presents alternative deep neural network architectures in order to reduce the computational complexity arising due to 3D-ConvNets used in all hybrid machine learning algorithms proposed in this work. These methodologies are exemplarily analysed with the HARRT algorithm, but the same methods can be used with other algorithms to reduce the required computational resources.

The outline of the chapter is as follows: Section 6.1 presents machine learning algorithms for replacing the computationally intensive analytical collision checking algorithm while Section 6.2 describes analytical methods to reduce the required computational resources by exploiting the design of machine learning and RRT algorithms. Section 6.3 presents the implementation procedure and measurement results for the HARRT algorithm on various hardware platforms using the optimization methodologies proposed in Section 6.1 and 6.2. Finally, Section 6.4 describes alternative machine learning architectures to 3D-ConvNet and compares them with each other based on the training results in terms of the top-2 accuracy.

6.1 Machine Learning Algorithms for Collision Checking

The ARRT and ARRT+ algorithms requires a module, which provides information on whether the vehicle will collide with any of the predictions of other road traffic participants along the planned trajectory. This module is called continuously with every new EGO vehicle state $\boldsymbol{s}(t)$ found while growing the tree $\mathcal{T}$.

In order to reduce the time needed for this module, some lazy planning algorithms have been proposed to delay collision checking until it iss needed [BK00, DSA13, KKY+16]. These algorithm check the collision only once a trajectory connecting the start position to the goal position is found. Once a collision is detected along the planned trajectory, the colliding segment is removed and the planning is continued. This methodology of lazy planning is extended in this work by replacing the computationally intensive modules for collision checking with machine learning algorithms until a trajectory is planned. Then the analytical methods are used to check the validity of the planned trajectory.

6.1.1 Machine Learning Based Collision Checking

As explained in Chapter 3, different computationally intensive analytical algorithms such as collision detection and calculation of the severity of injury, f_{road} and f_{obj}, along with a vehicle dynamic model are used in each prediction time-step to generate a tree $\mathcal{T}$ with many branches, i. e., many safe trajectories as shown in the Fig. 6.1. As only one safe trajectory (e. g. red) is selected for the vehicle to follow, the computation time required for finding other branches is wasted. Also, the number of times the functions f_{road} and f_{obj} are calculated increases with the number of time steps required to find the final trajectory. Also, within one call of these function, the collision check is performed for each collision object and the road boundaries in the EGO surrounding. In summary, the collision check is performed thousands of times for finding a collision-free trajectory of a few seconds. Therefore, machine learning algorithms $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1) \mapsto \hat{\mathbf{y}}_1$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2) \mapsto \hat{\mathbf{y}}_2$ are proposed for replacing the functions f_{obj} and f_{road}, respectively. The vectors $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ are the parameters to learn by minimizing the cross-entropy loss function between the labelled

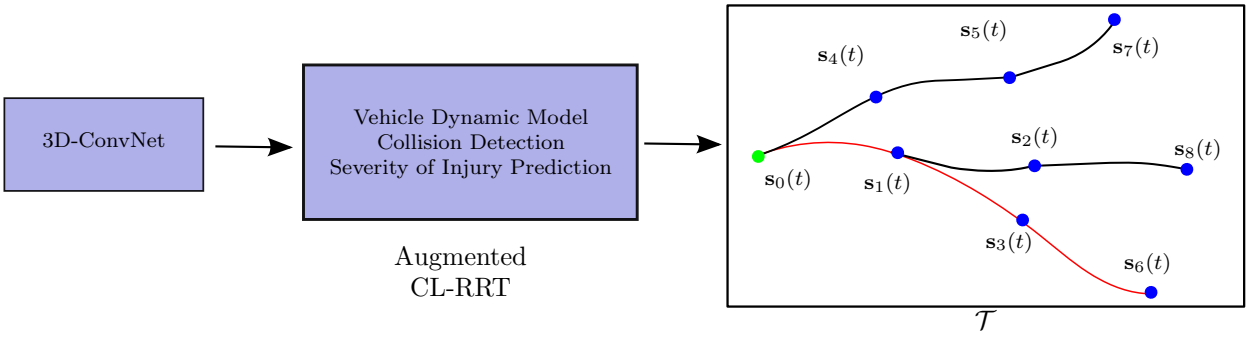


Figure 6.1: Tree $\mathcal{T}$ With Multiple Collsion-Free Trajectories.

and estimated posterior probabilities of the output. The vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the input feature vectors for two machine learning algorithms. Although implementing the collision checks with machine learning algorithms results into a minor reduction in the computation time in one run, due to the algorithms are called thousands of times for generating the tree, this results in a significant reduction in the required computation time. Once a safe trajectory is selected, it is rechecked with analytical algorithms for f_{obj} and f_{road}. This way, the considerable computation time for analytical algorithms is not wasted on branches of the tree $\mathcal{T}$, which will not be part of the final selected trajectory.

6.1.2 Feature and Labels for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$

The input features for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ are $\boldsymbol{x}_1 = [\mathbf{x}_{obj}^{\mathrm{T}}, \mathbf{y}_{obj}^{\mathrm{T}}, v_r, \mathtt{t}_{obj}]$, where $\{\mathbf{x}_{obj}, \mathbf{y}_{obj}\}$ are the X- and Y-coordinates of predictions of the four corners of the collision object in the body coordinate frame of the EGO vehicle that is in the state $\boldsymbol{s}(t)$, v_r is the relative velocity between the EGO vehicle and the collision object, and $\mathtt{t}_{obj}$ is the type of the collision object, i. e., a pedestrian, bicyclist or vehicle. Here, the position of the EGO vehicle is one of the positions $\boldsymbol{s}(t)$ along the branches of the tree $\mathcal{T}$ whose validity is to be checked. Similarly, the input features for $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$ consist of only a fixed number of X- and Y-coordinates of the nearest road outer line points $\boldsymbol{x}_{road}, \boldsymbol{y}_{road}$ from the center of gravity of the vehicle in the body coordinate frame of the EGO and the EGO velocity v, i. e., $\boldsymbol{x}_2 = [\mathbf{x}_{road}^{\mathrm{T}}, \mathbf{y}_{road}^{\mathrm{T}}, v]$. Going outside of the road is considered as colliding with a stationary objects and therefore, the EGO velocity is considered as the relative collision velocity for calculating the severity of injury. Fig. 6.2 shows an example of extracted features $[\mathbf{x}_{obj}, \mathbf{y}_{obj}]$ and $[\mathbf{x}_{road}, \mathbf{y}_{road}]$. The outputs $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are vectors representing three labels, *no collision*, *predicted nonsevere collision*, and *predicted severe collision*. When growing a tree $\mathcal{T}$ for trajectory planning, the states for which the predicted outputs of either the function $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$ are *predicted severe collision* are discarded. Otherwise, the states are added to the tree $\mathcal{T}$. The Matlab neural network toolbox [DB93] is used for training these networks.

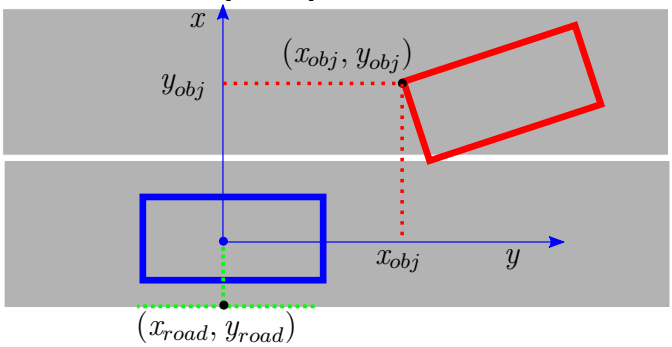


Figure 6.2: Features for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$.

6.1.3 Data Generation

The training samples for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ are generated by randomly sampling different object locations $[\mathbf{x}_{obj}^{\mathrm{T}}, \mathbf{y}_{obj}^{\mathrm{T}}]$ in the body coordinate frame of the EGO vehicle and the relative

velocity v_r. The output labels are found by using the function f_{obj}. The coordinates $[\mathbf{x}_{obj}^{\mathrm{T}}, \mathbf{y}_{obj}^{\mathrm{T}}]$ are randomly sampled such that the center of objects lie only within 10 meters range from the center of the EGO body coordinate frame as this is a range that is sufficient to eliminate the need for the collision detection. Similarly, training samples for $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$ are generated by using different road infrastructures, i. e., different road structures like curved roads, straight roads, intersections, etc., EGO velocity v and EGO positions. The labels for each training sample can be easily evaluated in the simulation.

The machine learning algorithm $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ performs the collision detection with each object in sequence while $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$ is trained with all types of road infrastructures. Therefore, they continue to adhere to the HARRT and HARRT+ algorithms principle of using one algorithm for all types of traffic-scenarios irrespective of the number and type of traffic participants.

6.1.4 Accuracy for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$

Using the above data generation procedure, 2 million and 1 million training samples are generated for the machine learning functions $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$, respectively. For both algorithms, neural networks with one hidden layer are used for training. 70% of the total data is used for training and 15% of the data is used for both validation and testing. The results of training with a different number of neurons in the hidden layer for both algorithms is shown in Table 6.1.

Table 6.1: Training Results (% accuracy) for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$.

	$\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$		$\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$	
# of neurons	**Validation Data**	**Training Data**	**Validation Data**	**Training Data**
10	93.3	93.2	97.3	98.7
20	98.2	98.1	99.8	99.7
30	97.7	97.6	99.3	97.3

A neural network with 20 neurons in the hidden layer is selected for implementation as it gives best results with the validation and training data of both machine learning algorithms as shown in the confusion matrices in Fig. 6.3 and Fig. 6.4. The class labels 1, 2, 3 in Fig. 6.3 and Fig. 6.4 are for *no collision*, *nonsevere collision*, and *severe collision*, respectively.

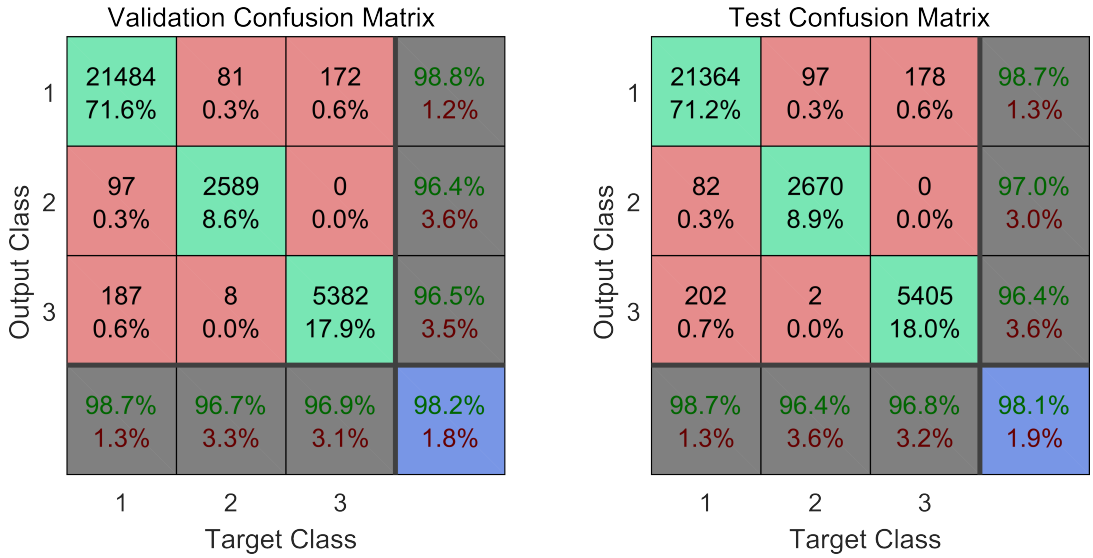


Figure 6.3: Confusion Matrices for Validation and Training Data of $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$.

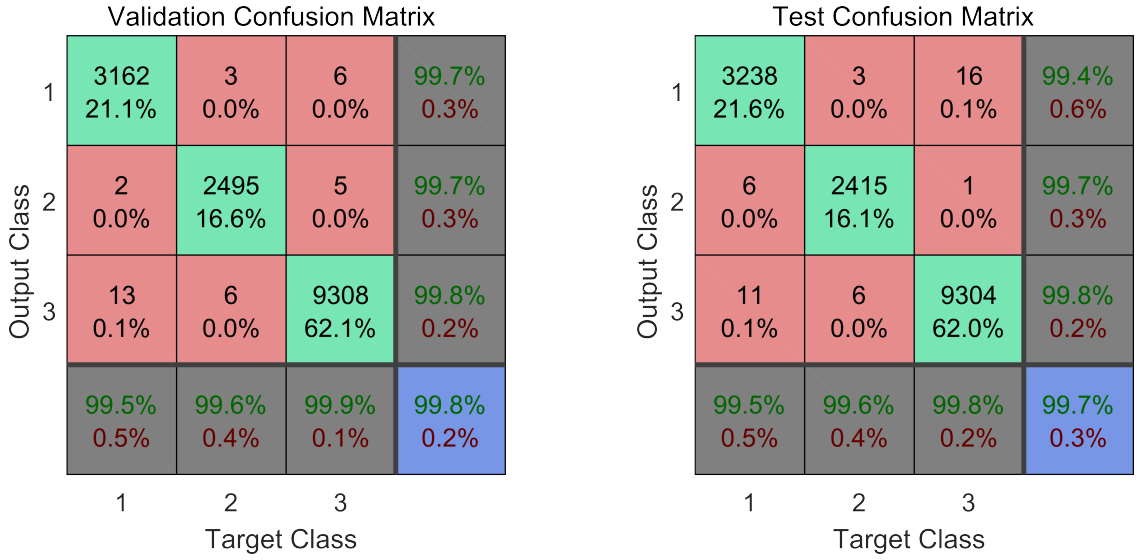


Figure 6.4: Confusion Matrices for Validation and Training Data of $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$

6.2 Analytical Algorithms for Reducing the Memory (SRAM)

For safety-critical applications like trajectory planning in complex traffic-scenarios, a dynamic memory allocation should be avoided [Hol06], as mentioned in one of the requirements for safety-critical application in Section 1.3. The memory required for the 3D-ConvNets, having an architecture as shown in the Fig. 5.5, will be very high if memory is

preallocated for the input $\mathbf{M}$, all weights $\mathbf{W}$, and feature maps $\mathbf{V}$ in all the convolution and pooling layers. Also, the processing in the cloud is not desirable due to latency, security, and communication bandwidth concerns. Therefore, different analytical algorithms are proposed to reduce the memory requirements for different parts of 3D-ConvNet as well as the ARRT algorithm. Sections 6.2.1, 6.2.2, 6.2.3 are the methodologies for memory reduction of 3D-ConvNets and Section 6.2.4 is the methodology for reducing the memory for generated trees $\mathcal{T}$ using the ARRT algorithm.

6.2.1 Iterative ConvNet Features Generation

A 3D-ConvNet architecture comprises of multiple 3D convolutions and 3D pooling layers generating multiple output feature maps as shown in the Fig. 5.5. In a 3D convolution layer, the value of the output unit ${}_kV^{xy}$ at position (x, y) and time t in the k^{th} out of K output feature maps is obtained by

$$ {}_kV^{xyt} = \sigma\left({}_kb + \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1}\sum_{n=0}^{k_3-1} {}_kW_{ijn}U_{ijn}^{xyt}\right), \tag{6.1} $$

where, k_1, k_2, and k_3 are the dimensions of the all K kernels ${}_kW$ and U_{ijn}^{xyt} is the input at position $(x + i\Delta x, y + j\Delta y, t + n\Delta t)$. The nonlinear function σ is the ReLU activation function. Similarly, the value of unit ${}_lV^{xyt}$ in the l^{th} out of L output feature maps of a 3D average pooling layer at position (x, y) and time t is obtained by

$$ {}_lP^{xyt} = \frac{\sum_{i=0}^{k_1'-1}\sum_{j=0}^{k_2'-1}\sum_{n=0}^{k_3'-1} {}_lV_{ijn}^{xyt}}{k_1'k_2'k_3'}, \tag{6.2} $$

where ${}_lV_{ijn}^{xyt}$ is l^{th} out of total L inputs for the pooling layer at position $(x + i\Delta x, y + j\Delta y, t + n\Delta t)$ and k_1', k_2', and k_3' are the dimensions of slice of the input over which the pooling is performed.

Eq. 6.1 shows that for each input for a 3D convolution, the number of output feature maps is equal to the number of kernels K, while Eq. 6.2 shows that the number of output feature maps in the pooling layer is equal to the number of input feature maps. As it can be seen in Fig. 5.5, for a single input $\mathbf{M}$, 8 feature maps are generated in layer C1 using 8 kernels ${}_k\boldsymbol{W}_{C1}, k = 1, 2, \ldots, 8$, followed by a pooling layer P1 which has also 8 output feature maps. For each output feature map from the P1 layer, further 8 output feature maps are generated with 8 kernels ${}_k\boldsymbol{W}_{C2}, k = 1, 2, \ldots, 8$, making in total 64 output feature maps for C2 and also 64 for the P2 layer. Preallocation of memory for all of these feature maps and kernels will require a huge amount of memory, which is not available in automotive microcontrollers.

The ConvNet features for the fully connected layer FC are obtained just by the vectorization of the output feature maps of the layer P2. The Calculation of all the feature maps in all layers at once for the inference is not necessary. Instead, they can be calculated iteratively

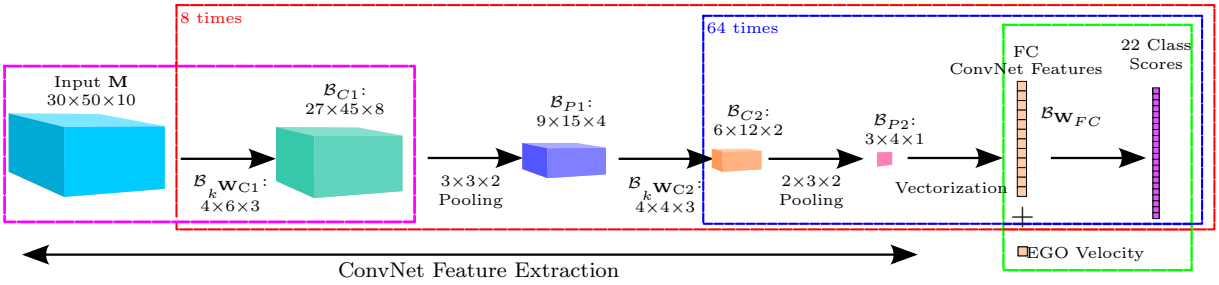


Figure 6.5: Modified Architecture of 3D-ConvNets

and appended to get the ConvNet Feature for the FC layer. Therefore, buffers $\mathcal{B}_{C1}$, $\mathcal{B}_{P1}$, $\mathcal{B}_{C2}$, and $\mathcal{B}_{P2}$ are created for the layers C1, P1, C2, and P2, respectively. These buffers designed according to the dimension for a feature map in respective layers. Kernels are used in sequence to calculate one feature map at the end of the layer P2, so buffers $\mathcal{B}_{k}\mathbf{w}_{C1}$ and $\mathcal{B}_{k}\mathbf{w}_{C2}$ are also defined for the kernels ${}_{k}\mathbf{W}_{C1}$ and ${}_{k}\mathbf{W}_{C2}$, respectively. These iteratively generated output feature maps can be vectorized and appended continuously to get the ConvNet Features for the FC layer. Fig. 6.5 shows this operational change for calculating the ConvNet Features in sequential manner with two nested iterations in the red box with kernels ${}_{k}\mathbf{W}_{C1}$ and the blue box for kernels ${}_{k}\mathbf{W}_{C2}$. If parallel processing is used to speed up the computation of the features, the buffers equal to the number of parallel processors can be created, which helps to increase the computation speed without assigning memory to all feature maps.

Table 6.2 shows the difference between the iterative and non-iterative ConvNet Features generation in terms of the required number of parameters to be stored, and the size of memory required in the convolution and pooling layers of the 3D-ConvNet.

Table 6.2: Difference Between Non-iterative and Iterative ConvNet Feature Generation

Layer	**Non-iterative method**		**Iterative method**	
	# Parameters	**Memory (kilobytes)**	**# Parameters**	**Memory (kilobytes)**
C1	77760	303.75	9720	37.97
P1	4320	16.88	540	2.11
C2	9216	36	144	0.563
P2	768	3	12	0.047
Total	92064	359.63	10416	40.69

6.2.2 Iterative Convolution of the Input M

As explained in Section 5.3.1, the input **M** is a sequence of occupancy grids having either the value 0 or 1 for each cell. Therefore, its data type can be defined as 'unsigned int'.

However, the kernels used for the convolution operation have the data type 'float' and this requires the input must also be 'float'. Even if the input is defined as 'unsigned int', the compiler changes the data type to 'float' before performing a convolution operation. As the input contains 15000 (dimension of $30 \times 50 \times 10$) parameters in total, this results in a huge memory consumption. As per Eq. 6.1, each unit of a output feature map in the C1 layer of the 3D-ConvNet is calculated iteratively with a kernel ${}_k\mathbf{W}_{\mathrm{C1}}$ convolving over a part of the input $\mathbf{M}$ equal to the dimension of the kernel ${}_k\mathbf{W}_{\mathrm{C1}}$. It offers a possibility to define the input $\mathbf{M}$ as 'unsigned int' and only copy part of the input into a buffer $\mathcal{B}_{\mathbf{M}}$ with equal dimension to the kernel ${}_k\mathbf{W}_{\mathrm{C1}}$ with data type 'float'. Therefore, instead of using 15000 'float' parameters, 15000 'unsigned int' parameters and a buffer $\mathcal{B}_{\mathbf{M}}$ of dimension equal to kernel ${}_k\mathbf{W}_{\mathrm{C1}}$ with data type 'float' are used. This iterative convolution of the input $\mathbf{M}$, shown in the part of the 3D-ConvNet with the magenta box in Fig. 6.5, results in a saving of memory equal to 43.66 kilobytes. Apart from this, as there are multiple repeated convolution operations with the same input patch containing all ones or zeros, the computation time for convolutions is reduced by reusing these convolution results.

6.2.3 Fully-Connected Layer in Compressed Sparse Column Format

The largest number of parameters of the 3D-ConvNet are in the fully-connected layer FC. The 768 ConvNet features are generated in sequence due to the use of iterative ConvNet features generation explained in Section 6.2.1. At every iteration of this procedure, only 12 ConvNet features are generated, which can be appended continuously to the previously found ConvNet features previously. The weight matrix $\mathbf{W}_{FC}$ having the dimension of 22×769 (16918 parameters) is multiplied with a vector of dimension 769×1, consisting of the generated ConvNet features and the EGO velocity, to get a vector of dimension 22×1 followed by the addition of 22 bias parameters as shown in the Fig. 6.6a. This vectort is further used as an input for a softmax function to generate 22 class scores. Thus, the total number of parameters in the FC layer is 16940. Although these parameters are stored in the FLASH memory of the microcontroller, they are fetched to SRAM during the calculation. Fetching all 16940 parameters at once requires a lot of memory in SRAM. Therefore, the multiplication of $\mathbf{W}_{FC}$ and ConvNet features is done in a Compressed Sparse Column (CSC) format [DRM14] as shown in Fig. 6.6b.

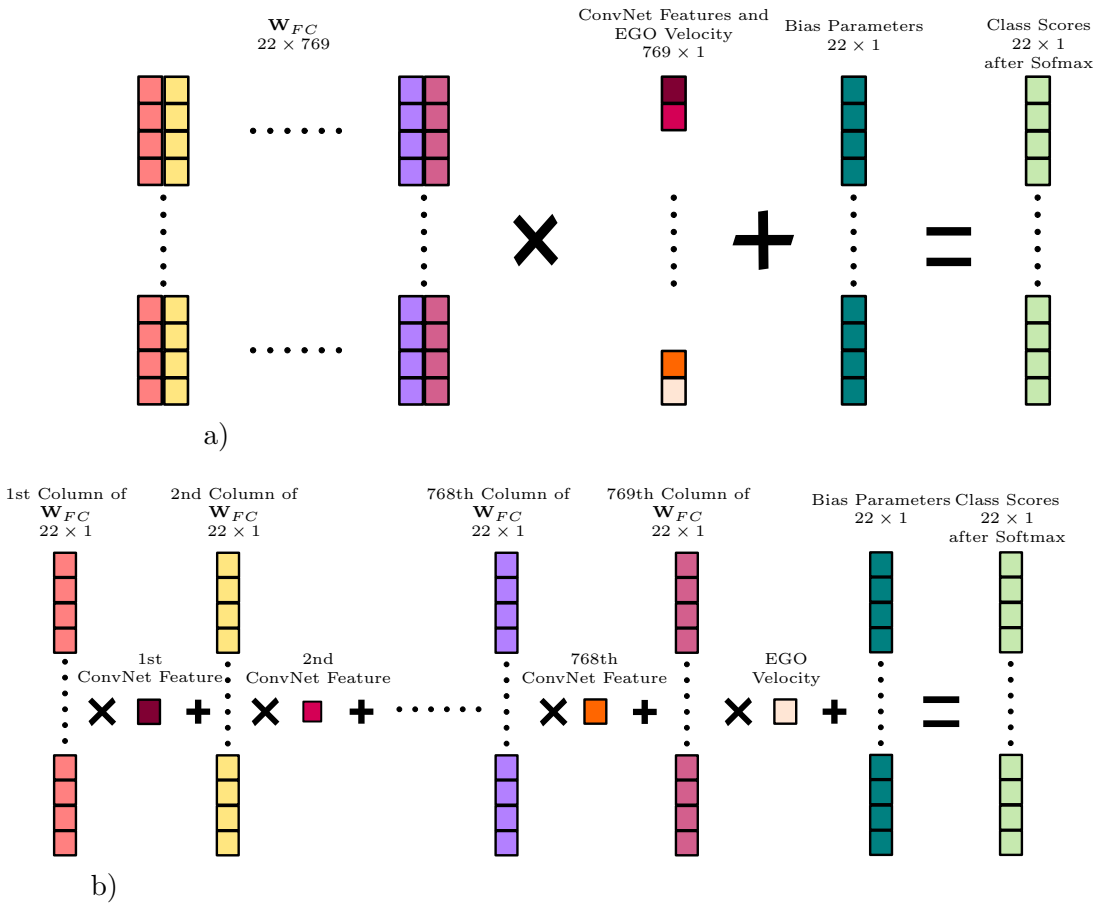


Figure 6.6: Difference Between Normal Fully-Connected Layer and Compressed Sparse Column Format.

The output of the product is calculated by taking iteratively the parameters in a single column of $\mathbf{W}_{FC}$ at a time in the buffer $\mathcal{B}_{\mathbf{W}_{FC}}$ of dimension 22×1 and multiplying it with the corresponding ConvNet feature which is added to the output continuously to get the class scores. This part of the 3D-ConvNet architecture is shown as a green box in the Fig. 6.5. It eliminates the requirement for fetching all 16940 parameters at once but requires memory equivalent to 22 parameters for the buffer $\mathcal{B}_{\mathbf{W}_{FC}}$ and 22 bias parameters saving memory for 16896 parameters. Also, as a single ConvNet feature is used in each iteration, finding all 768 ConvNet features at once is not required. Using the iterative ConvNet feature generation 12 features of the FC layer are iteratively generated as explained in Section 6.2.1. Therefore, the buffer $\mathcal{B}_{FC}$ of dimension 12×1 is used to store these features. It also reduces the required number of parameters by 756. Thus, usage of the CSC format multiplication in the FC layer saves memory equivalent to a total of 17652 parameters having data type 'float', i. e., 68.95 kilobytes.

6.2.4 Storage of Only One State with Low Severity of Injury

The states in the HARRT algorithm are still added to the tree $\mathcal{T}$ even if they are not collision-free, if the predicted severity of injury at these states is low. It provides an option of following a trajectory with predicted low severity of a collision, when a collision-free trajectory is not found at the cost of increasing the number of states to be stored in the tree, thereby increasing the required SRAM memory. The total number of states in the tree has to be predefined to preallocate the memory. Storing each state with the predicted low severity of collision can fill the memory quickly and the algorithm will not be able to store more collision-free states, even if computation time is available. To avoid this, only one state with a predicted low severity of the collision is stored. If a new state with even lower predicted severity of injury is found, then the previously stored state is just replaced.

6.3 Implementation Results in Hardware Platforms

For the implementation of the algorithms, a TMS570LS series Texas Instrument microcontroller TMS570LS20216, a dSPACE MicroAutobox II and a Raspberry Pi 3 are used. TMS570LS20216 is an automotive microcontroller certified for the use in IEC 61508 SIL3 safety systems. It has an ARM Cortex-R4F floating-point CPU, which runs with a speed of 160 MHz and has 2 MB and 160 KB of FLASH and SRAM size, respectively. The small amount of SRAM available in TMS570LS20216 microcontroller indicates the significance of the amount of memory reduction obtained by the methodologies described in Section 6.2. The dSPACE MicroAutobox II is a real-time system for performing fast function prototyping. It runs with the speed of 900 MHz and 16 MB RAM while the Raspberry Pi 3 has a frequency 1.2 GHz and 1 GB RAM. The results for the mex-implementation in Matlab are also presented. The computer used for the mex-implementation measurements has an Intel Core-i7 processor with 2.8 GHz and 8GB RAM.

The Simulink coder is used to generate the optimized C-code of the Matlab implementation of the ARRT, HARRT and optimized HARRT with machine learning and analytical methods proposed in Sections 6.1 and 6.2. This generated code is downloaded to TMS570LS20216, dSPACE MicroAutobox II, Raspberry Pi 3 and the required amount of memory (SRAM) and time for execution is measured. The results are summarized in Tables 6.3 and 6.4.

The results show that the computation time for mex-implementation and dSPACE MicroAutobox is very low, but the computation time for the automotive microcontroller TMS570LS20216 is still quite high. With parallel processing, many times reductionss is possible as there is massive scope for parallelization in different parts of the algorithms such as in predictions of road objects, 3D-ConvNet as well as in the RRT algorithms. Many approaches are being proposed [PZP17, HZC+17] to make the implementation of convolutional neural networks efficient. Also, another efficient approach [NBS17] with machine learning methods for prediction of occupancy grids can be used to reduce the computation

Table 6.3: Required Computational Time in Milliseconds for Trajectory Planning Algorithms.

Method	**Mex**	**TMS570**	**MA II**	**Rasp.-Pi**
ARRT	28.7410	–	340	800
HARRT	12.5790	–	98	290
+ $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$	10.1994	–	47	220
+ Iterative ConvNetFeatures Generation	8.9062	–	36	200
+ Iterative Convolution of the Input $\mathbf{M}$	8.2747	–	19	140
+ FC Layer in CSC Format	8.2747	423	19	140

Table 6.4: Required Computational Memory for Trajectory Planning Algorithms

Method	**SRAM (KB)**
Augmented CL-RRT	374.01
Hybrid Augmented CL-RRT	812.3
+ $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$	590.89
+ Iterative ConvNet Features Generation	271.50
+ Iterative Convolution of the Input $\mathbf{M}$	227.84
+ FC Layer in CSC Format	158.893

time.

6.4 Alternative Network Architectures

3D-ConvNet is chosen for extracting features from the input $\mathbf{M}$ as it is a suitable model for extracting the required spatio-temporal features. However, this network requires a high computational memory. Section 6.1 proposes a few methodologies for reducing the required computational memory. An alternative approach to reduce the required computational memory is to choose a different computationally efficient network architecture, which also learns the spatio-temporal features from the input $\mathbf{M}$.

In 3D ConvNet, the 3D feature maps are generated by using 3D filters, having a smaller depth dimension than its input depth. These feature maps are not appended like in 2D ConvNet. In fact, the second convolutional layer performs convolution on each generated 3D feature map individually, as shown in Fig. 5.5. This affects the performance of the network as the spatial feature correlation in different feature maps at a particular time step is not learned in intermediate layers. Only in the fully-connected layer, all the extracted features are combined. However, the fully-connected layer surrenders all the spatio-temporal

correlation information in the input. Therefore, the used architecture might not be capable enough and an analysis of other possible architectures is needed. The deep learning community have proposed many alternative architectures [PC14, LFV+16] for processing videos. Taking those approaches as motivation, this section proposes few alternative architectures that need comparatively low computational resources and do not surrender the spatio-temporal feature correlation as well.

The recurrent convolutional neural network (Recurrent-CNN) [PC14], as shown in Fig. 6.7, is a natural selection for learning spatio-temporal features due its suitable architecture. The weight matrix $\boldsymbol{W}_{xh}$ represents weights of 2D-ConvNet, which will learn the spatial features from input occupancy grids while the connection of latent spaces using the weights $\boldsymbol{W}_{hh}$ will learn the temporal correlation of these spatial features. RNNs share the weights at each step and the extraction of features from inputs at different time-steps can be parallelized, which makes them highly computationally efficient. Another advantage for RNNs is that they do not need a fixed size of the input. As the number of time-steps (10) is small, the complex LSTM or GRU networks, which eliminate the problem of vanishing and exploding gradient, are not required.

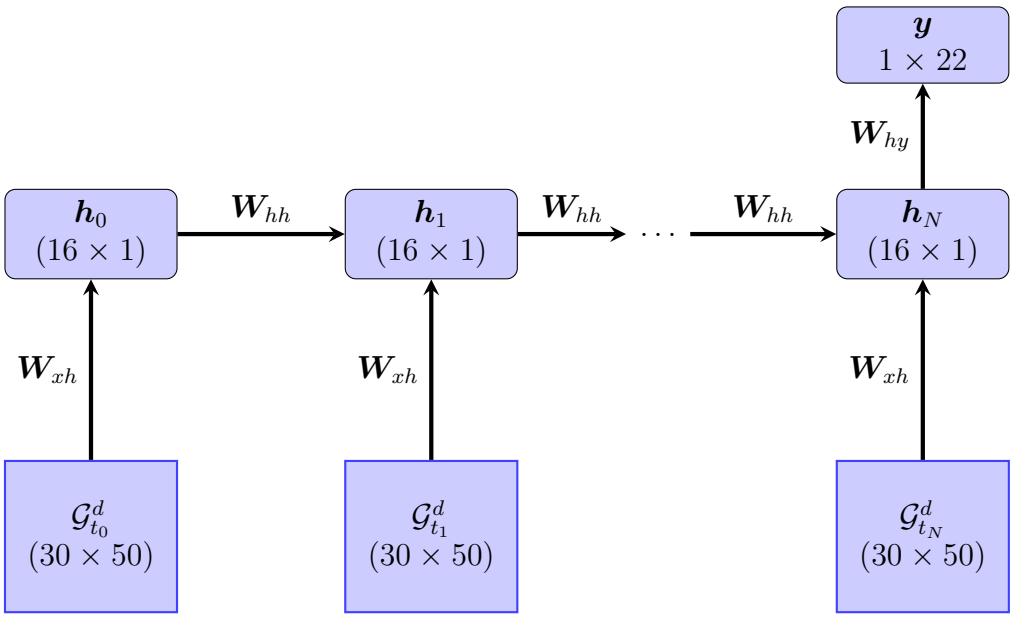


Figure 6.7: Recurrent Neural Network.

With the input data structure $\mathbf{M}$ having a fied length of time dimension, it is not necessary to use RNN as well. An alternative methodology is to extract initially only spatial features from an individual occupancy grid of $\mathbf{M}$ using simple 2D-ConvNet and then combining those features in a single vector followed by a fully-connected layer that predicts the label. The weights of 2D-ConvNet can be shared equivalently to the RNN architecture. This is

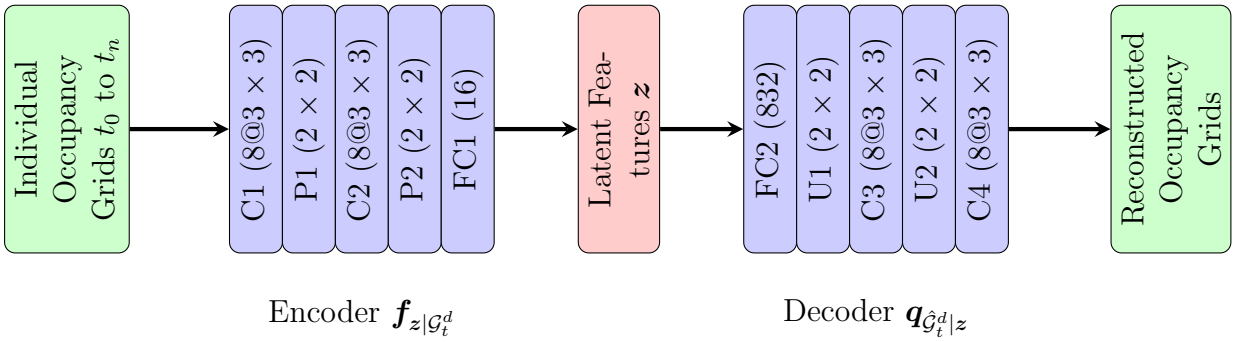


Figure 6.8: Autoencoder for Extracting Features $\boldsymbol{z}$.

achieved by initially training an autoencoder with individual occupancy grids of all data sample D as input, as shown in Fig. 6.8. The encoder $\boldsymbol{f}_{\boldsymbol{z}|\mathcal{G}_t^d}$, where $t \in \{t_0, \ldots, t_n\}$ and $d \in \{1, \ldots, D\}$, is used as a common 2D-ConvNet. The problem with this methodology is that although a 2D-ConvNet learns the spatial feature correlation, the correlation of the temporal features is surrendered only in the fully-connected layer. This problem can be solved by using the 1D-TCN [LFV+16] architecture.

Temporal convolutional neural networks (TCN) is a family of networks for convolutional sequence prediction. A temporal convolution using 1D filters is applied in [LFV+16] to capture how the input signals evolve over the course of action for video-based action recognition. Dilated causal convolutions are used in [ODZ+16] for the raw audio generation to learn long-range temporal dependencies. A convolutional encoder network with soft attention followed by the LSTM decoder is used in [GAGD17] for language translation. A gated convolutional network is used for language modeling in [DFAG17]. All these approaches are for long time series. As the time-steps in the given problem are fixed and short, the 1D-TCN followed by a fully-connected layer is used, as shown in Fig. 6.9.

The weights $\boldsymbol{W}_{hh}$ in RNN represent the same architecture as the encoder $\boldsymbol{f}_{\boldsymbol{z}|\mathcal{G}_t^d}$. The total number of learnable parameters (weights and biases) for 1D-TCN architecture is 13992. Table 6.5 shows the number of weights in each layer of the encoder.

Table 6.5: Learnable Parameters of Encoder $\boldsymbol{f}_{\boldsymbol{z}|\mathcal{G}_t^d}$ or $\boldsymbol{W}_{hh}$ in RNN.

Layer	**Number of Parameters**
C1	80
P1	0
C2	584
P2	0
FC1	13328
Total	13992

Table 6.6 provides the comparison between all the presented networks in this section based

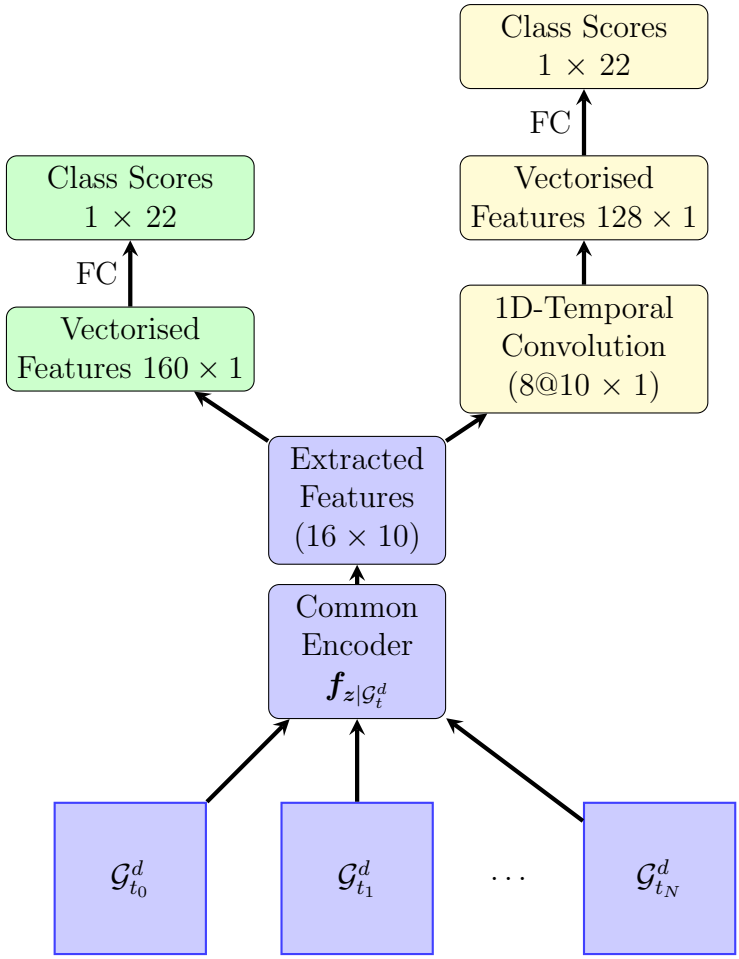


Figure 6.9: FC and 1D-TCN Architecture Networks.

on the accuracy in terms of the top-2 error and the total number of parameters. The models were trained on the data generated as described in 5.3.1. For FC, 1D-TCN and RNN models, the number of parameters is the sum of parameters in the encoder network and the parameters from the rest of the network. There is no significant difference in the number of parameters in all networks. However the 1D-TCN network provides the best accuracy on test data.

Table 6.6: Comparison Between Architectures.

	3D-CNN	**FC**	**1D-TCN**	**RNN**
-top-2 (%) error	9.32	11.36	**7.75**	9.69
# parameters	17916	13992+3542	13992+2926	**13992 + 646**

Summary

In the first part of this chapter, some machine learning and analytical approaches for the reduction of computational complexity of the hybrid machine learning algorithms along with their practical implementation in different embedded hardwares are presented. The second part of this chapter dealt with the comparison of different machine learning architectures with 3D-ConvNet based on the accuracy.

Chapter 7

Conclusion

This thesis presents hybrid machine learning methods, which are a combination of machine learning methods and physical models, for safety-critical application with vehicle trajectory planning for critical traffic-scenarios as a primary example. The basic idea of the combination is to predict an approximate solution with machine learning algorithms, which is used as a reference to find the final solution with a vehicle dynamic model. In the first part of this thesis, the motivation for vehicle trajectory planning in critical traffic-scenarios along with the aim and proposed methodology are briefly stated. It also lists the general requirements for vehicle trajectory planning, considering it as a safety-critical application. The second part of this thesis deals with dynamic models and sampling-based trajectory planning algorithms. The dynamic models and controllers for road traffic-participants are presented in Chapter 2. The validation procedure for vehicle dynamic models in this chapter shows the nonlinear two-track vehicle dynamic model outputs accurate trajectories also with harsh control actions. Therefore, this model is chosen to be used in the sampling-based trajectory planning algorithms, namely *Augmented CL-RRT* (ARRT) and *Augmented CL-RRT+* (ARRT+), described in Chapter 3. These algorithms are extended from the already proposed algorithm called CL-RRT in order to make them suitable for critical traffic-scenarios with multiple static and dynamic objects. These algorithms also include a methodology for estimating the severity of injury so that the trajectories can be selected to mitigate a collision in case a collision-free trajectory cannot be found. In the third part of this thesis, the focus shifts towards machine learning. Chapter 4 illustrates the basics of machine learning, along with the theory of machine learning algorithms used in this thesis. Then, different hybrid machine learning algorithms are proposed in Chapter 5. Firstly, it presents the *Hybrid Augmented CL-RRT* (HARRT) and *Hybrid Augmented CL-RRT+* (HARRT+) algorithms, which use 3D convolutional neural networks to bias the sampling strategies of ARRT and ARRT+ algorithms, respectively. Secondly, it also explains the algorithm *Generative Algorithm for Trajectory Exploration* (GATE), which is a deterministic optimization-based trajectory planning algorithm in combination with a generative model for vehicle trajectories. This algorithm is further combined with the

ARRT+ algorithm and named as GATE-ARRT+. The simulation results with multiple critical traffic-scenarios show that hybrid machine learning algorithms are more efficient than their corresponding sampling-based algorithms without compromising safety. In the final chapter, some machine learning and analytical approaches are described to reduce the required computational resources for the proposed hybrid machine learning algorithms. The implementation results in various hardware platforms show the real-time capability of the proposed algorithms.

7.1 Comparative Analysis of Trajectory Planning Algorithms

All the trajectory planning algorithms proposed in this thesis use different sampling or planning strategies that might be effective in certain types of traffic-scenarios, but at the same time, theyy can be adverse in other traffic-scenarios for the convergence of the algorithm. Although the different comparisons using simulation results, presented in Chapter 5, show that generally hybrid machine learning algorithms are efficient compared to their corresponding model-based approaches, a detailed analysis is needed to understand the suitability of all these algorithms in different types of traffic-scenarios.

In general, the complexity of the traffic-scenario is positively correlated with the number of road traffic-participants. Therefore, the number of road traffic-participants is changed in curved road traffic-scenarios. Many critical curved road traffic-scenarios are simulated and safe trajectories are found using the ARRT, HARRT, ARRT+, HARRT+, GATE and GATE-ARRT+ algorithm. The generated results are shown in Tables 7.1, 7.2, 7.3. The second column in these tables present the ARRT algorithm that uses only a constant velocity profile, i. e., no longitudinal dynamic intervention. Therefore, this column is marked as ARRT(0) The third column indicates details of the ARRT algorithm the uses 21 different predefined acceleration profiles. The limit for the number of samples that can be used for the ARRT(21) and ARRT+ is set to 2100 while for the ARRT(0), HARRT, HARRT+, GATE-ARRT+ the limit is 300 samples. These results are generated in the Matlab simulation environment explained in Section 2.4 with the implementation of algorithms without any computational memory or time optimization approaches described in the Chapter 6 as the purpose of this analysis is not to measure the final required computation time but only performing the comparative analysis of different trajectory planning algorithms described in this work.

The following observations can be made from the simulation results:

- The ARRT(0) algorithm finds a collision-free trajectory in a drastically smaller number of scenarios compared to other trajectory planning algorithms. The reason for this ineffectiveness is that many critical traffic-scenarios need a simultaneous lateral and longitudinal dynamic intervention for the EGO vehicle to avoid the collision.

Table 7.1: Comparison of Trajectory Planning Algorithms: Curved Road Scenarios with 3 Dynamic Objects (Total Traffic-Scenarios: 738).

	ARRT (0)	ARRT (21)	HARRT	ARRT+	HARRT+	GATE	GATE-ARRT+
Collision-free Trajectory Found	276	691	671	674	652	712	692
No Safe Trajectory Found	174	11	19	12	28	11	18
Average Time (sec.)	1.1221	6.9251	1.1502	6.8428	1.3782	0.3412	0.4931
Standard Deviation	0.2823	0.7952	0.3759	0.8123	0.2068	0.0713	0.1482
Worst-Case Time	2.2798	8.9376	2.9342	9.8984	1.9744	1.1532	1.4223

Table 7.2: Comparison of Trajectory Planning Algorithms: Curved Road Scenarios with 4 Dynamic Objects (Total Traffic-Scenarios: 1047)

	ARRT (0)	ARRT (21)	HARRT	ARRT+	HARRT+	GATE	GATE-ARRT+
Collision-free Trajectory Found	226	801	772	821	843	734	929
No Safe Trajectory Found	771	19	28	27	43	69	25
Average Time (sec.)	0.9361	6.3662	0.855	6.9589	1.1267	0.8117	0.7103
Standard Deviation	0.2457	1.4651	0.2552	0.9397	0.3257	0.3721	0.2131
Worst-Case Time	2.4072	12.4313	3.0185	11.4504	2.0035	1.3724	1.532

- The HARRT and HARRT+ algorithms not only reduce the average execution time but also the variance of execution time compared to their corresponding model-based trajectory planning algorithms, which is an essential requirement for random sampling-based search algorithms as mentioned in Section 1.3.
- The average computation time and its variance for finding safe trajectories with all trajectory planning algorithms increase with the number of objects.

Table 7.3: Comparison of Trajectory Planning Algorithms: Curved Road Scenarios with 5 Dynamic Objects (Total Traffic-Scenarios: 4275).

	ARRT (0)	ARRT (21)	HARRT	ARRT+	HARRT+	GATE	GATE-ARRT+
Collision-free Trajectory Found	1127	3334	3276	3862	3819	2282	3770
No Safe Trajectory Found	1771	25	41	69	91	134	38
Average Time (sec.)	1.3703	7.0826	1.3092	7.3710	1.7773	1.2132	0.9321
Standard Deviation	0.7144	2.7881	0.8269	2.7577	1.0175	0.3231	0.7341
Worst-Case Time	2.4564	24.0242	4.4709	24.5066	4.3418	1.6312	3.7213

- With the increase in the number of objects, the available free-space reduces. Therefore, the ARRT+ algorithm, which has more randomness than the ARRT algorithm, can find collision-free trajectories in more traffic-scenarios.

- In scenarios where no collision-free trajectory can be found, the deterministic nature of the ARRT algorithm of using a strong braking acceleration profile is more useful than the random sampling strategy of the ARRT+ algorithm.

- The GATE algorithm converges rapidly and consistently in traffic-scenarios with few road traffic-participants as more free-space is available and being a fully deterministic algorithm, there is virtually no variance in the required computational time.

- The GATE-ARRT+ algorithm, provides the best results in comparison to other hybrid machine learning algorithms. Only the GATE algorithm is better with traffic-scenarios having only three objects.

In summary, this comparative analysis shows that the deterministic algorithms give better performance in traffic-scenarios with a low number of road traffic-participants. In contrast, the effectiveness of hybrid machine learning algorithms with sampling-based algorithms increases with the number of road traffic-participants, i. e., as the complexity of traffic-scenarios increases. The scenarios in which no safe trajectory can be found, a combination of deterministic full braking with random sampling for the lateral dynamic intervention prove to be the best strategy for finding a trajectory with a low severity of injury.

7.2 Contributions and Future Scope

The main contribution of this work is the methodology of hybrid machine learning algorithms, a combination of machine learning and physical models, for safety-critical application. The basic idea behind the combination is to find an approximate solution with machine learning algorithms, which are used as a reference to find a final solution with dynamic models. Although the primary application for this method presented is safe trajectory planning in critical traffic-scenarios, it is also suitable for other safety-critical applications as well.

Specifically, this work presents the hybrid machine learning algorithms HARRT and HARRT+ combined with two sampling-based trajectory planning algorithms with vehicle dynamic constraints ARRT and ARRT+, respectively. These algorithms also include a methodology for estimating the severity of injury that enables choosing a trajectory to mitigate an unavoidable collision. The work also introduces a hybrid machine learning algorithm GATE that combines a generative model for trajectories with the deterministic optimization-based algorithm for planning safe trajectories. This algorithm is further combined with the ARRT+ algorithm and named as GATE-ARRT+. Finally, this work proposes multiple machine learning algorithms and analytical procedures for the optimization of computing resources required by these algorithms so that they can be implemented in a highly resource-constrained automotive microcontroller and run in real-time.

An exciting field for future work is to adapt the hybrid machine learning approach in a suitable format for probabilistic planning with a reinforcement learning framework. The similarity and dissimilarity in the proposed and reinforcement learning algorithms are already described in Chapter 5. The proposed algorithms make some assumptions, e. g., that the modern sensors provide complete information about the vehicle surrounding or that the single hypothesis prediction for other road traffic-participants. The reinforcement learning algorithms for solving POMDPs (partially observable Markov decision processes) with probabilistic predictions of road traffic-participants is an interesting alternative for the algorithms proposed in this thesis.

This work presents multiple sampling-based algorithms with different sampling strategies. Although these strategies show positive results, there are still many possibilities available. In particular, the criteria for defining the ratio of biased and random sampling is still understudied. This work suggests to increase the percentage of random sampling with the complexity of the scenario and the complexity of the traffic-scenario is correlated with only the number of road traffic-participants and road infrastructure. Therefore, a detailed study to understand the complexity of traffic-scenarios with dynamic objects is necessary. Also, the convergence rate of hybrid machine learning algorithms depends on the quality of the solution provided by machine learning algorithms as a reference. If the traffic-scenario is very similar to the traffic-scenario in the training data for the machine learning algorithm, then it will provide a better solution, which means a biased sampling should be preferred and vice versa. Therefore, a traffic-scenario comparison methodology is another criteria

for defining the ratio of biased and random sampling. Apart from this, adaptive sampling strategies are scarce in trajectory planning algorithms, which makes it a very interesting field for future work.

This thesis proposes hybrid machine learning methods only for a specific application, i. e., safe trajectory planning in critical-traffic scenarios. However, these methods can be easily modified for trajectory planning in other domains such as medical robots, industrial automation, etc. The basic idea of the combination of machine learning algorithms for finding an initial solution and then converging to the final solution with physical models is applicable for any field with two possible aims: safety and low computational resources.

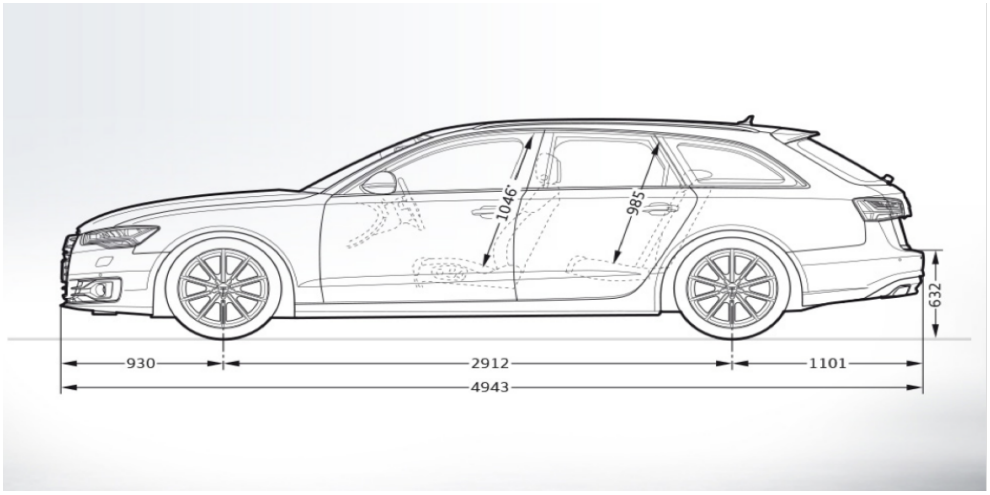
Chapter 8

Appendix

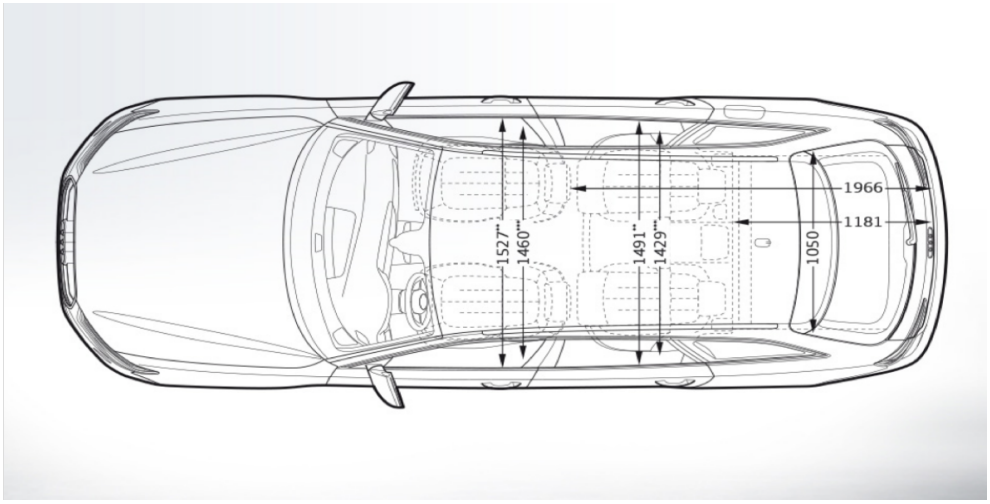
8.1 Technical Specifications for Audi A6 Avant

Table 8.1: Audi A6 Avant Technical Specifications.

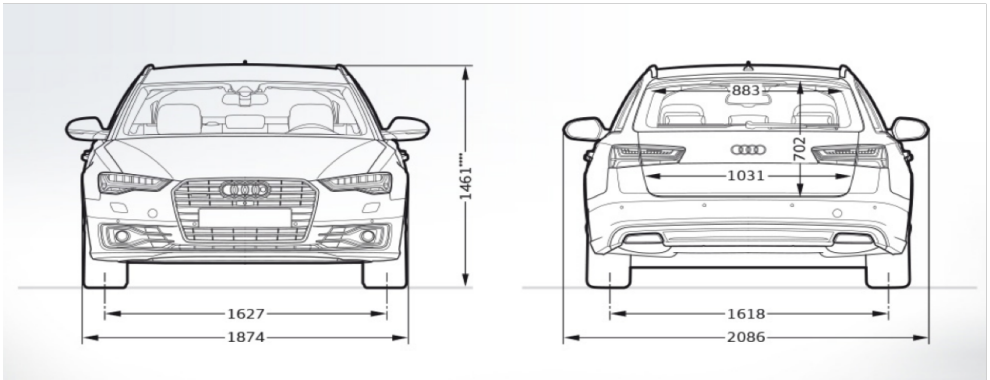
Engine	
Capacity	2967 cm^3
Maximum Power	230 kW
Maximum Torque	2800 Nm
Highest Speed	250 km/h
Acceleration 0-100 km/h	5.4 s
Weight	
Net Weight (According to Datasheet)	1978
Total Weight (with Full Tank)	2485
Weight Measurements While Testing	
Front Right	600 kg
Front Left	568 kg
Rear Left	599 kg
Rear Right	545 kg



a) Side View



b) Top View



c) Front and Rear View

Figure 8.1: The Dimension of Audi A6 Avant from Different Views

8.2 Backpropagation in Neural Networks

This section explains the *theory of backpropagation*, which is the way of computing gradients through recursive application of the chain rule in neural networks. The goal of backpropagation is to compute the partial derivatives of the cost function E with respect to any weight (or bias) in the neural network. In other words, backpropagation is about understanding how changing the weights and biases in a network changes the cost function. This is important to understand, and effectively, to develop and design neural networks. This section describes some basic terms necessary to understand the backpropagation algorithm, followed by the backpropagation procedure for simple feedforward neural networks, convolutional neural networks and recurrent neural networks.

8.2.1 Interpretation of the Derivative

The derivative of a function with a real variable measures the sensitivity of its output value with respect to its input values, i. e., it defines how much the output value will change with an infinitesimal change in the input. As an example, the derivative of the function $f(x)$ is computed as

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{8.1}$$

There can be multiple parameters, instead of only one, on which function output value depends. The concept of partial derivatives is used in such cases. A partial derivative of a function of several variables is its derivative with respect to one of those variables, with the other variables held constant. For example, the partial derivatives $\frac{\partial f}{\partial x_1}$ and $\frac{\partial f}{\partial x_2}$ for a function $f(x_1, x_2) = x_1 x_2$ with respect to variables x_1 and x_2 are

$$\frac{\partial f}{\partial x_1} = x_2, \frac{\partial f}{\partial x_2} = x_1. \tag{8.2}$$

The vector of these partial derivatives is called the *gradient of function* $f(x_1, x_2)$, denoted as $\nabla f(x_1, x_2)$ and expressed as

$$\nabla f(x_1, x_2) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right] = [x_2, x_1]. \tag{8.3}$$

8.2.2 Chain Rule

A function can be composed of further multiple functions and not just variables. Such a function is called as *composite function*. It is indicated as $f(g(x))$, where f is a function of the function $g(x)$. The chain rule describes the derivative of such composite functions as

$$\frac{d}{dx} [f(g(x))] = f'(g(x))g'(x). \tag{8.4}$$

This can also be written as

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}. \tag{8.5}$$

In order to understand the practical use of chain rule, consider a simple composite function $f(x_1, x_2, x_3) = (x_1+x_2)x_3$. By defining a new function $g = (x_1+x_2)$, the function f becomes $f = gx_3$. The partial derivative of the function f with respect to variables x_1 using the chain rule is x_3 as $\frac{df}{dg} = x_3$ and $\frac{dg}{dx_1} = 1$. Similarly, the partial derivatives of function f with respect to variables x_2 and x_3 can be calculated as x_3 and $(x_1 + x_2)$. Therefore, the gradient of function f is $\nabla f = [x_3, x_3, x_1 + x_2]$. This methodology is used to calculate the derivative of any composite functions by breaking them into simpler functions whose derivative is easy to calculate or already known.

8.2.3 Vector Function Derivative

In order to understand the vector function derivative, consider the simple equation

$$\boldsymbol{f} = \boldsymbol{W}\boldsymbol{x}, \tag{8.6}$$

where $\boldsymbol{f} \in \mathbb{R}^M$, $\boldsymbol{W} \in \mathbb{R}^{M\times N}$ and $\boldsymbol{x} \in \mathbb{R}^N$. The derivative of $\boldsymbol{f}$ with respect to $\boldsymbol{x}$ is a $M \times N$ Jacobian matrix

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix}. \tag{8.7}$$

The individual i^{th} element f_i in the vector $\boldsymbol{f}$ can be expressed as

$$\begin{aligned} f_i &= \sum_{n=1}^{N} W_{i,n}x_j, \\ &= W_{i,1}x_1 + W_{i,2}x_2 + \ldots + W_{i,N}x_N. \end{aligned} \tag{8.8}$$

The partial derivative of the i^{th} element of $\boldsymbol{f}$ with respect to the j^{th} element in $\boldsymbol{x}$ is expressed as

$$\frac{\partial f_i}{\partial x_j} = 0 + 0 + \ldots + W_{i,j} + \ldots + 0 = W_{i,j}. \tag{8.9}$$

Using the above expression, all the elements in the Jacobian matrix in Eq. 8.7 can be replaced and the derivative of $\boldsymbol{f}$ with respect to $\boldsymbol{x}$ becomes

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} W_{1,1} & \cdots & W_{1,N} \\ \vdots & \ddots & \vdots \\ W_{M,1} & \cdots & W_{M,N} \end{bmatrix} = \boldsymbol{W}. \tag{8.10}$$

8.2.4 Backpropagation in Feedforward Neural Networks

The feedforward neural network is also a composite function comprised of many functions represented by layers and activation functions. Therefore, it is possible to apply the chain rule to neural networks as long as individual operations in the network are derivable. The

general function representing a layer in neural network having total L layers as per Eq, 4.20 is

$$\boldsymbol{f}^l = \boldsymbol{\sigma}^l(\boldsymbol{W}^l \boldsymbol{f}^{l-1} + \boldsymbol{b}^l), \tag{8.11}$$

where $\boldsymbol{W}^l$ is the weighted connection between the l^{th} and $(l-1)^{th}$ layer, $\boldsymbol{b}^l$ is the vector of bias parameters in the l^{th} layer and $\boldsymbol{\sigma}^l$ is the nonlinear activation function. The intermediate quantity $(\boldsymbol{W}^l \boldsymbol{f}^{l-1} + \boldsymbol{b}^l)$ calculated for the calculation of $\boldsymbol{f}^l$, is named as $\boldsymbol{z}^l$:

$$\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{f}^{l-1} + \boldsymbol{b}^l. \tag{8.12}$$

Therefore, the Eq. 8.11 can also be written as

$$\boldsymbol{f}^l = \boldsymbol{\sigma}^l(\boldsymbol{z}^l). \tag{8.13}$$

In order to compute these partial derivatives, an intermediate quantity, δ_j^l, which is called as the *neuron error*[1] in the j^{th} neuron of l^{th} layer, is first calculated. This is defined as

$$\delta_j^l = \frac{\partial E}{\partial z_j^l}. \tag{8.14}$$

This expression indicates how a small change in z_j^l would affect the cost function E. If $\frac{\partial E}{\partial z_j^l}$ is high, then the cost function can be reduced by choosing ∂z_j^l such that the sign of $\frac{\partial E}{\partial z_j^l}$ is changed. By contrast, if $\frac{\partial E}{\partial z_j^l}$ has a value close to zero, the neuron is already near-optimal state and it cannot contribute any more in decreasing the cost function. Thus, the expression δ_j^l is intuitively a measure of the error in the neuron. Therefore, the term δ_j^l is defined as the neuron error.

To reiterate, the aim of the backpropagation procedure is to compute partial derivatives $\frac{\partial E}{\partial W_{j,k}^l}$ and $\frac{\partial E}{\partial b_j^l}$ for the weights and biases in all $l = 1, 2, \ldots, L$ layers. The index j and k here denote the number of the neuron in l^{th} and $(l-1)^{th}$ layer to which the respective weight and bias parameters are associated. The steps in backpropagation are as follows:

- Set the input $\boldsymbol{x}$ as the first activation $\boldsymbol{f}^1$.
- The feedforward computation of the network for each layer $l = 2, 3, \ldots, L$, i. e., compute Eq. 8.12 and Eq. 8.13.
- Calculate output neuron error $\boldsymbol{\delta}^L$.
- Backpropagate neuron error $\boldsymbol{\delta}^l$ for each layer $l = L-1, L-2, \ldots, 2$.
- Calculate the gradients $\frac{\partial E}{\partial W_{j,k}^l}$ and $\frac{\partial E}{\partial b_j^l}$

[1] The term error here has quite different meaning than its meaning in the classification tasks which is classification failure rate.

After performing the feedforward computation, the next step is the computation of output neuron error $\boldsymbol{\delta}^L$. According to the Eq. 8.14 , the j^{th} component of the neuron error in the output layer, $\boldsymbol{\delta}^L$, is given as

$$\delta_j^L = \frac{\partial E}{\partial z_j^L}. \tag{8.15}$$

Applying the chain rule, the equation changes to

$$\delta_j^L = \sum_k \frac{\partial E}{\partial f_k^L} \frac{\partial f_k^L}{\partial z_j^L}, \tag{8.16}$$

where the sum is over all neurons in the output layer. $\frac{\partial f_k^L}{\partial z_j^L}$ is zero for all $k \neq j$. Therefore the Eq. 8.16 simplifies to

$$\delta_j^L = \frac{\partial E}{\partial f_j^L} \frac{\partial f_j^L}{\partial z_j^L}. \tag{8.17}$$

From Eq. 8.13, the second term on the right side in the above equation can be replaced by $\sigma'(z_j^L)$ and the equation becomes

$$\delta_j^L = \frac{\partial E}{\partial f_j^L} \sigma'(z_j^L). \tag{8.18}$$

The neuron error can be written in matrix form as

$$\boldsymbol{\delta}_L = \nabla_f \boldsymbol{E} \odot \boldsymbol{\sigma}'(\boldsymbol{z}^L). \tag{8.19}$$

Often, the error function used is the cross-entropy loss function

$$E = -\sum_k f_k^L \log \hat{f}_k^L, \tag{8.20}$$

where f_k^L and $\hat{f}_k^L$ are target and predicted labels. Therefore, Eq. 8.16 can be written as

$$\begin{aligned} \delta_j^L &= -\sum_k f_k^L \frac{\partial \log \hat{f}_k^L}{\partial \hat{f}_k^L} \frac{\partial \hat{f}_k^L}{\partial z_j^L} \\ &= -\sum_k \frac{f_k^L}{\hat{f}_k^L} \frac{\partial \hat{f}_k^L}{\partial z_j^L} \end{aligned} \tag{8.21}$$

The function $\hat{f}_k^L$ is often the softmax function. Therefore, the term $\frac{\partial \hat{f}_k^L}{\partial z_j^L}$ is given as

$$\frac{\partial \hat{f}_k^L}{\partial z_j^L} = \begin{cases} -\hat{f}_k^L \hat{f}_j^L, & k \neq j, \\ \hat{f}_k^L (1 - \hat{f}_k^L), & k = j. \end{cases} \tag{8.22}$$

Putting Eq. 8.21 and Eq. 8.22 together, we get

$$\begin{aligned}\delta_j^L &= -\frac{f_j^L}{\hat{f}_j^L}\hat{f}_j^L(1-\hat{f}_j^L) + \sum_{k\neq l}(-\frac{f_k^L}{\hat{f}_k^L})(-\hat{f}_k^L\hat{f}_j^L) \\ &= -f_j^L + f_j^L\hat{f}_j^L + \sum_{k\neq l}\hat{f}_k^L\hat{f}_j^L \\ &= -f_j^L + \hat{f}_j^L\sum_k f_k^L\end{aligned} \tag{8.23}$$

As f_k^L, the target label, is defined as one-hot vector, then the sum just equals to 1 and we get a simple expression for the output neuron error as

$$\delta_j^L = \hat{f}_j^L - f_j^L. \tag{8.24}$$

The next step is the core of the backpropagation algorithm in which the neuron error $\boldsymbol{\delta}^L$ is backpropagated to previous layers, i. e., the calculation of neuron errors δ_j^l for $l = (L-1), (L-2), \ldots, 2$. In general, the neuron error is backpropagated stepwise from l^{th} to $(l-1)^{th}$ layer. Therefore, the Eq. 8.14 is rewritten in terms of $\delta_k^{(l+1)}$ using chain rule such that

$$\delta_k^l = \sum_k \frac{\partial E}{\partial z_k^{(l+1)}}\frac{\partial z_k^{(l+1)}}{\partial z_j^L} \tag{8.25}$$

The first term on the right side in the above equation is $\delta_k^{(l+1)}$. Substituting this and the expression for $z_k^{(l+1)}$ from Eq. 8.12 and 8.13., the equation changes to

$$\begin{aligned}\delta_k^l &= \sum_k \frac{\partial(\sum_j W_{j,k}^{(l+1)}\sigma(z_j^l))}{\partial z_j^l}\delta_k^{(l+1)} \\ &= \sum_k W_{j,k}^{(l+1)}\sigma'(z_j^l)\delta_k^{(l+1)}.\end{aligned} \tag{8.26}$$

This can be written in matrix form as

$$\boldsymbol{\delta}^l = ({\boldsymbol{W}^{l+1}}^T\boldsymbol{\delta}_k^{(l+1)}) \odot \sigma'(\boldsymbol{z}^l). \tag{8.27}$$

Once the neuron error is backpropagated till the first layer of the network, the next step is to calculate the gradients $\frac{\partial E}{\partial W_{j,k}^l}$ and $\frac{\partial E}{\partial b_j^l}$ again using the chain rule. The expression for $\frac{\partial E}{\partial W_{j,k}^l}$ with the chain rule is given as

$$\frac{\partial E}{\partial W_{j,k}^l} = \frac{\partial E}{\partial z_j^l}\frac{\partial z_j^l}{\partial W_{j,k}^l}. \tag{8.28}$$

The first term on the right side of the above equation is the neuron error δ_j^l. Substituting this and the expression for z_j^l from Eq. 8.12 the equation becomes

$$\begin{aligned}\frac{\partial E}{\partial W_{j,k}^l} &= \delta_j^l\frac{\partial(W_{j,k}^l f_k^{(l-1)}) + b_j^l}{\partial W_{j,k}^l} \\ &= \delta_j^l f_k^{(l-1)}.\end{aligned} \tag{8.29}$$

Basically, this equation suggests that the gradient of the weight is dependent on the activation of input neuron $f_k^{(l-1)}$ and the output neuron error δ_j^l. The consequence of this is that if the input activation is small, $f_k^{(l-1)} \approx 0$, then the gradient of the weight is also small. As a consequence of this the weight changes slowly during the gradient descent procedure which means weight outputs from low activations learn slowly. In similar way, the gradient for bias can be expressed using the chain rule as

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l. \tag{8.30}$$

8.2.5 Backpropagation in Convolutional Neural Networks

The convolutional neural network (ConvNet) consist of multiple convolution, pooling and fully connected layers. The backpropagation algorithm for fully connected layer is already described in the previous section. This section explains the backpropagation algorithm for convolution and pooling operation.

Convolution Layer

The equation for l^{th} layer out of L layers in ConvNet, which involves convolution operation, as per in Eq. 4.29 is

$$V_{xy}^l = \sigma\left(b^l + \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} W_{ij} V_{x+i,y+j}^{l-1}\right). \tag{8.31}$$

The intermediate quantity

$$Z_{xy}^l = b^l + \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} W_{ij}^l V_{x+i,y+j}^{l-1} \tag{8.32}$$

is defined so that

$$V_{xy}^l = \sigma(Z_{xy}^l). \tag{8.33}$$

Again, the aim of the backpropagation procedure is to find the partial derivatives $\frac{\partial E}{\partial W_{j,k}^l}$ and $\frac{\partial E}{\partial b_j^l}$. The output neuron error δ^L is calculated similary as in the feedforward neural network. This error is backpropagated to previous layers. This backpropagated error $\delta_{x,y}^l$ is a measure of how the change in pixel $Z_{x,y}^l$ affects the loss function E. Therefore, it is expressed as

$$\delta_{x,y}^l = \frac{\partial E}{\partial Z_{x,y}^l}. \tag{8.34}$$

Unlike feedforward neural networks, ConvNets have sparse connectivity. Therefore, the region affected by pixel $Z_{x,y}^l$ in the next layer $(l+1)$ is bounded by the region of $\boldsymbol{V}^{l+1}$ having the top left corner pixel and bottom right corner pixel at index $(x-k_1+1, y-k_2+1)$

and (x, y), respectively and denoted as Q. These pixels can also be defined A REGION using chain rule and sums,

$$\begin{aligned}\frac{\partial E}{\partial Z^l_{x,y}} &= \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} \frac{\partial E}{\partial Z^{l+1}_{x-i,y-j}} \frac{\partial Z^{l+1}_{x-i,y-j}}{\partial Z^l_{x,y}} \\ &= \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} \delta^{l+1}_{x-i,y-j} \frac{\partial Z^{l+1}_{x-i,y-j}}{\partial Z^l_{x,y}}.\end{aligned} \tag{8.35}$$

The ranges for summation iterators are $0 \leqslant i \leqslant k_1 - 1$ and $0 \leqslant j \leqslant k_2 - 1$ and the height and width of the region Q is defined as $x - i$ and $y - j$. The second term in Eq 8.35 can be expanded using the expression in Eq. 8.32

$$\begin{aligned}\frac{\partial Z^{l+1}_{x-i,y-j}}{\partial Z^l_{x,y}} &= \frac{\partial}{\partial Z^l_{x,y}} \left(b^{l+1} + \sum_{i'=0}^{k_1-1}\sum_{j'=0}^{k_2-1} W^{l+1}_{i'j'} V^l_{x-i+i',y-j+j'} \right) \\ &= \frac{\partial}{\partial Z^l_{x,y}} \left(b^{l+1} + \sum_{i'=0}^{k_1-1}\sum_{j'=0}^{k_2-1} W^{l+1}_{i'j'} \sigma\left(Z^l_{x-i+i',y-j+j'} \right) \right) \\ &= \frac{\partial}{\partial Z^l_{x,y}} \left(b^{l+1} + W^{l+1}_{i'j'} \sigma\left(Z^l_{x-i+0,y-j+0} \right) + \ldots + W^{l+1}_{i'j'} \sigma\left(Z^l_{x,y} \right) + \ldots \right) \\ &= \frac{\partial}{\partial Z^l_{x,y}} \left(W^{l+1}_{i'j'} \sigma\left(Z^l_{x,y} \right) \right) \\ &= W^{l+1}_{ij} \frac{\partial}{\partial Z^l_{x,y}} \left(\sigma\left(Z^l_{x,y} \right) \right) \\ &= W^{l+1}_{ij} \sigma'\left(Z^l_{x,y} \right)\end{aligned} \tag{8.36}$$

Substituting the above expression in Eq. 8.35

$$\begin{aligned}\frac{\partial E}{\partial Z^l_{x,y}} &= \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} \delta^{l+1}_{x-i,y-j} W^{l+1}_{ij} \sigma'\left(Z^l_{x,y} \right) \\ &= rot_{180^0} \left(\sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} \delta^{l+1}_{x+i,y+j} W^{l+1}_{ij} \right) \sigma'\left(Z^l_{x,y} \right) \\ &= \left(\boldsymbol{\delta}^{l+1}_{x,y} * rot_{180^0}(W^{l+1}) \right) \sigma'\left(Z^l_{x,y} \right).\end{aligned} \tag{8.37}$$

The convolution operation between input of size $v_1 \times v_2$ and kernel of size $k_1 \times k_2$ produces an output of size $(v_1 - k_1 + 1) \times (v_2 - k_2 + 1)$. The gradients for individual weight $W^l_{x,y}$ can be obtained by applying the chain rule in the following way:

$$\begin{aligned}\frac{\partial E}{\partial W^l_{x,y}} &= \sum_{i=0}^{v_1-k_1}\sum_{j=0}^{v_2-k_2} \frac{\partial E}{\partial Z^l_{i,j}} \frac{\partial Z^l_{i,j}}{\partial W^l_{x,y}} \\ &= \sum_{i=0}^{v_1-k_1}\sum_{j=0}^{v_2-k_2} \delta^l_{i,j} \frac{\partial Z^l_{i,j}}{\partial W^l_{x,y}}.\end{aligned} \tag{8.38}$$

The second term on the right hand side of the above equation can be expanded by substituting Eq. 8.32 such that

$$\begin{aligned}\frac{\partial Z^l_{i,j}}{\partial W^l_{x,y}} &= \frac{\partial}{\partial W^l_{x,y}}\left(b^l + \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1} W^l_{x,y} V^{l-1}_{i+x,j+y}\right) \\ &= \frac{\partial}{\partial W^l_{x,y}}\left(b^l + W^l_{0,0}V^{l-1}_{i+0,j+0} + \ldots + W^l_{x,y}V^{l-1}_{i+x,j+y} + \ldots\right) \\ &= 0 + \ldots + \frac{\partial}{\partial W^l_{x,y}}\left(W^l_{x,y}V^{l-1}_{i+x,j+y}\right) + \ldots \\ &= V^{l-1}_{i+x,j+y}.\end{aligned} \tag{8.39}$$

Combining Eq. 8.38 and Eq. 8.39, the gradient $\frac{\partial E}{\partial W^l_{x,y}}$ is given as

$$\begin{aligned}\frac{\partial E}{\partial W^l_{x,y}} &= \sum_{i=0}^{v_1-k_1}\sum_{j=0}^{v_2-k_2} \delta^l_{i,j} V^{l-1}_{i+x,j+y} \\ &= \delta^l_{i,j} * V^{l-1}_{x,y}.\end{aligned} \tag{8.40}$$

By performing similar computations, the gradient $\frac{\partial E}{\partial b^l}$ is computed as

$$\frac{\partial E}{\partial b^l} = \sum_{i=0}^{v_1-k_1}\sum_{j=0}^{v_2-k_2} \delta^l_{i,j}. \tag{8.41}$$

Pooling Layer

Although there are no learning parameters in a pooling layer, the error need to be backpropagated through this layer as well so that the error for previous convolutional layers can be calculated. In general, a pooling layer converts a block of size $k'_1 \times k'_2$ into a single unit from which the error is also backpropagated. The equations for max-pooling and average-pooling operations are rewritten as

$$V^{l+1}_{xy} = max(V^l_{x:x+k'_1,y:y+k'_2}), \tag{8.42}$$

$$V^{l+1}_{xy} = \frac{\sum_{i=0}^{k'_1-1}\sum_{j=0}^{k'_2-1} V^l_{x+i,y+j}}{k'_1 k'_2}, \tag{8.43}$$

respectively. The term $V^l_{x:x+k'_1,y:y+k'_2}$ in the Eq. 8.42 represents the array with elements in the block of feature map V^l with corners $x, x+k'_1, y, y+k_2$. In case of max-pooling, the error $\delta^{l+1}_{x,y}$ in the unit $V^{l+1}_{x,y}$ is completely assigned to the maximum element from the block $V^l_{x:x+k'_1,y:y+k'_2}$ as other units did not contribute anything to it. All other units are assigned zero error. Therefore, the index of the maximum unit need to be recorded during the

forward propagation. This error mathematically is expressed as

$$\begin{aligned}\frac{\partial E}{\partial V_{x,y}^{l}} &= \frac{\partial E}{\partial V_{x,y}^{l+1}} \frac{\partial V_{x,y}^{l+1}}{\partial V_{i,j}^{l}} \\ &= \delta_{x,y}^{l+1} \frac{\partial V_{x,y}^{l+1}}{\partial V_{i,j}^{l}}.\end{aligned} \tag{8.44}$$

The second term on right hand side of the above equation is calculated using the following relationship

$$\frac{\partial V_{x,y}^{l+1}}{\partial V_{i,j}^{l}} = \begin{cases} 1, & V_{i,j}^{l} = max(V_{x+0,y+0}^{l}, \ldots, V_{x+k_1,y+k_2}^{l}), \\ 0, & otherwise. \end{cases} \tag{8.45}$$

On the other hand, the error $\delta_{x,y}^{l+1}$ in the unit $V_{x,y}^{l+1}$ is multiplied with $\frac{1}{k_1' k_2'}$ and assigned to the whole block responsible for its calculation during forward propagation.

Nomenclature

B	body coordinate frame
C	center of the vehicle
D	dimensionality of the $\mathbf{x}$
D_{KL}	Kullback-Leiber Divergence
G	global coordinate frame
I	moment of inertia
K	the total number of classes
L	angular momentum
L	total number of longitudinal acceleration profiles
N	number of samples in the training dataset $\mathcal{D}$
S	a two dimension state-space
S_{free}	part of the state-space where the vehicle is free to move
S_{obs}	part of state-space occupied by the static and dynamic road traffic-participants
Δt	time interval for which the the tree is is extended in one iteration
α_M	hyperparameter for stochastic gradient descent with momentum
$\alpha_i j$	tire slip angle for the ij^{th} tire
α_{ROC}	scaling factor for ROC analysis
β	sideslip angle

$\boldsymbol{F}$	force
$\boldsymbol{M}$	moment
$\boldsymbol{W}$	weights of the neural network
$\boldsymbol{W}^l$	weights of the l^{th} layer of neural network
$\boldsymbol{\eta}$	EGO vehicle physical parameters
$\boldsymbol{a}$	acceleration
$\boldsymbol{r}$	position vector
$\boldsymbol{s}$	vehicle state
$\boldsymbol{s}_{goal}$	goal region for the safe trajectory
$\boldsymbol{s}_{init}$	initial state of the vehicle for the safe trajectory planning algorithm
$\boldsymbol{s}_{near}$	the nearest state in the tree from the sample $\boldsymbol{s}_{rand}$
$\boldsymbol{s}_{rand}$	the random sample
$\boldsymbol{u}$	control input to the the vehicle
$\boldsymbol{v}$	velocity
$\boldsymbol{x}$	realization of $\mathbf{x}$
$\boldsymbol{\omega}$	angular velocity
$\boldsymbol{\theta}$	parameters of machine learning algorithm
$\boldsymbol{\theta}_{ML}$	maximum likelihood estimation for parameters $\boldsymbol{\theta}$
$\mathbf{r}$	the reference input to the CL-RRT algorithm
$\mathbf{x}$	random variable representing the feature vector
$\boldsymbol{a}_{\mathbf{x}}$	longtudinal acceleration
δ	steering angle
δ_{ij}	wheel angle for theij^{th} tire

$\dot{\psi}$	yaw rate
ℓ	wheel-base
ℓ_{f}	distance between the center of gravity and front axle of the vehicle
ℓ_{r}	distance between the center of gravity and rear axle of the vehicle
ϵ_k	learning rate
λ	type of acceleration (positive, negative, zero or constant)
$\mathbf{M}$	the stack of predicted occupancy grids
$\mathcal{B}_{\{.\}}$	buffer memory allocated for {.}
$\mathcal{D}$	training dataset
$\mathcal{E}$	prediction risk
$\mathcal{E}_b$	bias error
$\mathcal{E}_v$	variance error
$\mathcal{E}_{emp}$	empirical prediction risk
$\mathcal{G}_t$	the predicted occupancy grid at time t
$\mathcal{L}(.,.)$	loss function
$\mathcal{T}$	the tree generated with RRT algorithm
$\mathcal{U}$	uniform distribution
I_{obj}	indicator function for detecting the collision with other road traffic-participants
I_{road}	indicator function for detecting if the vehicle state is out of the road
D	random variable for the training dataset
U	total potential
U_{att}	attractive potential
U_{rep}	repulsive potential

y	random variable representing the output label
μ	friction coefficient
$\nabla_{\boldsymbol{\theta}}$	gradient of parameters $\boldsymbol{\theta}$
ψ	yaw angle of the vehicle
σ	activation function in the neural network
τ	total time interval for which the safe trajectory is planned
τ_1	time interval for which the RRT algorithm plans the trajectory
v	magnitude of the velocity vector $\boldsymbol{v}$
c_k	the k^{th} class
c_ℓ, b_ℓ, c_s, b_s	constant tire parameters for the magic tire formula
g	acceleration due to gravity
m	mass
p	linearmomentum
p_{data}	data distribution
s_ℓ	tire longitudinal slip
s_{lij}	lateral slip for theij^{th} tire
s_{lij}	longitudinal slip for theij^{th} tire
s_s	tire lateral slip
t	time
t_0	initial time-step from which the safe trajectory for the vehicle is planned
t_c	time duration after t_0 at which the collision is predicted
w	track-width
y	realization of y

List of Symbols

B	body coordinate frame
C	center of the vehicle
D	dimensionality of the $\mathbf{x}$
D_{KL}	Kullback-Leiber Divergence
G	global coordinate frame
I	moment of inertia
K	the total number of classes
L	angular momentum
L	total number of longitudinal acceleration profiles
N	number of samples in the training dataset $\mathcal{D}$
S	a two dimension state-space
S_{free}	part of the state-space where the vehicle is free to move
S_{obs}	part of state-space occupied by the static and dynamic road traffic-participants
Δt	time interval for which the the tree is is extended in one iteration
α_M	hyperparameter for stochastic gradient descent with momentum
$\alpha_i j$	tire slip angle for the ij^{th} tire
α_{ROC}	scaling factor for ROC analysis
β	sideslip angle

$\boldsymbol{F}$	force
$\boldsymbol{M}$	moment
$\boldsymbol{W}$	weights of the neural network
$\boldsymbol{W}^l$	weights of the l^{th} layer of neural network
$\boldsymbol{\eta}$	EGO vehicle physical parameters
$\boldsymbol{a}$	acceleration
$\boldsymbol{r}$	position vector
$\boldsymbol{s}$	vehicle state
$\boldsymbol{s}_{goal}$	goal region for the safe trajectory
$\boldsymbol{s}_{init}$	initial state of the vehicle for the safe trajectory planning algorithm
$\boldsymbol{s}_{near}$	the nearest state in the tree from the sample $\boldsymbol{s}_{rand}$
$\boldsymbol{s}_{rand}$	the random sample
$\boldsymbol{u}$	control input to the the vehicle
$\boldsymbol{v}$	velocity
$\boldsymbol{x}$	realization of $\mathbf{x}$
$\boldsymbol{\omega}$	angular velocity
$\boldsymbol{\theta}$	parameters of machine learning algorithm
$\boldsymbol{\theta}_{ML}$	maximum likelihood estimation for parameters $\boldsymbol{\theta}$
$\mathbf{r}$	the reference input to the CL-RRT algorithm
$\mathbf{x}$	random variable representing the feature vector
$\boldsymbol{a}_{\mathbf{x}}$	longtudinal acceleration
δ	steering angle
δ_{ij}	wheel angle for theij^{th} tire

$\dot{\psi}$	yaw rate
ℓ	wheel-base
ℓ_{f}	distance between the center of gravity and front axle of the vehicle
ℓ_{r}	distance between the center of gravity and rear axle of the vehicle
ϵ_k	learning rate
λ	type of acceleration (positive, negative, zero or constant)
$\mathbf{M}$	the stack of predicted occupancy grids
$\mathcal{B}_{\{.\}}$	buffer memory allocated for {.}
$\mathcal{D}$	training dataset
$\mathcal{E}$	prediction risk
$\mathcal{E}_b$	bias error
$\mathcal{E}_v$	variance error
$\mathcal{E}_{emp}$	empirical prediction risk
$\mathcal{G}_t$	the predicted occupancy grid at time t
$\mathcal{L}(.,.)$	loss function
$\mathcal{T}$	the tree generated with RRT algorithm
$\mathcal{U}$	uniform distribution
I_{obj}	indicator function for detecting the collision with other road traffic-participants
I_{road}	indicator function for detecting if the vehicle state is out of the road
D	random variable for the training dataset
U	total potential
U_{att}	attractive potential
U_{rep}	repulsive potential

y	random variable representing the output label
μ	friction coefficient
$\nabla_{\boldsymbol{\theta}}$	gradient of parameters $\boldsymbol{\theta}$
ψ	yaw angle of the vehicle
σ	activation function in the neural network
τ	total time interval for which the safe trajectory is planned
τ_1	time interval for which the RRT algorithm plans the trajectory
v	magnitude of the velocity vector $\boldsymbol{v}$
c_k	the k^{th} class
c_ℓ, b_ℓ, c_s, b_s	constant tire parameters for the magic tire formula
g	acceleration due to gravity
m	mass
p	linearmomentum
p_{data}	data distribution
s_ℓ	tire longitudinal slip
s_{lij}	lateral slip for theij^{th} tire
s_{lij}	longitudinal slip for theij^{th} tire
s_s	tire lateral slip
t	time
t_0	initial time-step from which the safe trajectory for the vehicle is planned
t_c	time duration after t_0 at which the collision is predicted
w	track-width
y	realization of y

List of Figures

Bibliography

[Ahn] Song Ho Ahn. Convolution. http://www.songho.ca/dsp/convolution/convolution.html. Accessed: 2018-12-14.

[Ami18] Chaulwar Amit. Trafficsim, 2018.

[AS11] B. Akgun and M. Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2640–2645, Sept 2011.

[AT13] O. Arslan and P. Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *2013 IEEE International Conference on Robotics and Automation*, pages 2421–2428, May 2013.

[ATB17] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5360–5370. Curran Associates, Inc., 2017.

[AW96] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 113–120 vol.1, April 1996.

[BBS13] M. Brunner, B. Brüggemann, and D. Schulz. Hierarchical rough terrain motion planning using an optimal sampling-based method. In *2013 IEEE International Conference on Robotics and Automation*, pages 5539–5544, May 2013.

[BBSL94] Xuan-Nam Bui, J. . Boissonnat, P. Soueres, and J. . Laumond. Shortest path synthesis for dubins non-holonomic robot. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2–7 vol.1, May 1994.

[BIS09] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[BK00] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 521–528 vol.1, April 2000.

[Bot17] Michael Botsch. Lecture notes to the lecture "integrated safety and assistance systems", 2017.

[BV03] James Bruce and Manuela M. Veloso. Real-time randomized path planning for robot navigation. In Gal A. Kaminka, Pedro U. Lima, and Raúl Rojas, editors, *RoboCup 2002: Robot Soccer World Cup VI*, pages 288–295, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BYC+17] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence D. Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *CoRR*, abs/1704.07911, 2017.

[CAHBU19] Amit Chaulwar, Hussein Al-Hashimi, Michael Botsch, and Wolfgang Utschick. Efficient hybrid machine learning algorithm for trajectory planning in critical traffic-scenarios. pages 196–202, 09 2019.

[CAHBU20] Amit Chaulwar, Hussein Al-Hashimi, Michael Botsch, and Wolfgang Utschick. Sampling algorithms combination with machine learning for efficient safe trajectory planning. *International Journal of Machine Learning and Computing*, 1(1):1, 1 2020. Accepted.

[Cau47] Augustin-Louis Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte Rendu des S'eances de L'Acad'emie des Sciences XXV*, S'erie A(25):536–538, October 1847.

[CBKM16] Amit Chaulwar, Michael Botsch, Torsten Krüger, and Thomas Miehling. Planning of safe trajectories in dynamic multi-object traffic-scenarios. 2016.

[CBU16] A. Chaulwar, M. Botsch, and W. Utschick. A hybrid machine learning approach for planning safe trajectories in complex traffic-scenarios. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 540–546, Dec 2016.

[CBU17] A. Chaulwar, M. Botsch, and W. Utschick. A machine learning based biased-sampling approach for planning safe trajectories in complex, dynamic traffic-scenarios. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 297–303, June 2017.

[CBU18] Amit Chaulwar, Michael Botsch, and Wolfgang Utschick. Generation of reference trajectories for safe trajectory planning. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis, edi-

tors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 423–434, Cham, 2018. Springer International Publishing.

[CJ16] et. el. Chris Jurewicz. Exploration of vehicle impact speed – injury severity relationships for application in safer road design. *Transport Research Arena*, 14:4247–4256, 2016.

[Cou92] R. Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical Report CMU-RI-TR-92-01, Carnegie Mellon University, Pittsburgh, PA, January 1992.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[DB93] Howard Demuth and Mark Beale. Neural network toolbox for use with matlab – user's guide verion 3.0, 1993.

[DFAG17] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 933–941. JMLR.org, 2017.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[DK07] R. Diankov and J. Kuffner. Randomized statistical path planning. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–6, Oct 2007.

[Dom00] Pedro Domingos. A unified bias-variance decomposition for zero-one and squared loss. In *AAAI/IAAI*, pages 564–569. AAAI Press, 2000.

[dP18] Marcos Lopez de Prado. *Advances in Financial Machine Learning*. Wiley Publishing, 1st edition, 2018.

[DRM14] Richard Dorrance, Fengbo Ren, and Dejan Markovic. A scalable sparse matrix-vector multiplication kernel for energy-efficient sparse-blas on fpgas. pages 161–170, 02 2014.

[DSA13] J. Denny, K. Shi, and N. M. Amato. Lazy toggle prm: A single-query approach to motion planning. In *2013 IEEE International Conference on Robotics and Automation*, pages 2407–2414, May 2013.

[ESJ16] M. Elbanhawi, M. Simic, and R. Jazar. Randomized bidirectional b-spline parameterization motion planning. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):406–419, Feb 2016.

[FKS06] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. Replanning with rrts. In *ICRA*, pages 1243–1248. IEEE, 2006.

[FS06] D. Ferguson and A. Stentz. Anytime rrts. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5369–5375, Oct 2006.

[GAGD17] Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin. A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 123–135, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GO07] Roland Geraerts and Mark H. Overmars. Reachability-based analysis for probabilistic roadmap planners. *Robot. Auton. Syst.*, 55(11):824–836, November 2007.

[Gra13] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[GYZ+17] H. Gong, C. Yin, F. Zhang, Z. Hou, and R. Zhang. A path planning algorithm for unmanned vehicles based on target-oriented rapidly-exploring random tree. In *2017 11th Asian Control Conference (ASCC)*, pages 760–765, Dec 2017.

[HK00] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in prm planners. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 1408–1413 vol.2, April 2000.

[HL16] Jinwook Huh and D. D. Lee. Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–69, May 2016.

[HL18] J. Huh and D. D. Lee. Efficient sampling withq-learning to guide rapidly exploring random trees. *IEEE Robotics and Automation Letters*, 3(4):3868–3875, Oct 2018.

[HLK06] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *The International Journal of Robotics Research*, 25(7):627–643, 2006.

[HNR68] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[Hol06] Gerard Holzmann. The power of 10: Rules for developing safety-critical code. *IEEE Computer*, 39:95–97, 06 2006.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[HWL17] X. Han, T. Wang, and B. Liu. Path planning for robotic manipulator in narrow space with search algorithm. In *2017 2nd International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 339–344, Aug 2017.

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

[Jaz08] R.N. Jazar. *Vehicle Dynamics: Theory and Application*. Engineering (Springer-11647; ZDB-2-ENG). Springer, 2008.

[Jok93] H C Joksch. Velocity change and fatality risk in a crash–a rule of thumb. *Accident; analysis and prevention*, 25 1:103–4, 1993.

[JP13] Lucas Janson and Marco Pavone. Fast marching trees: A fast marching sampling-based method for optimal motion planning in many dimensions. In *Robotics Research - The 16th International Symposium ISRR, 16-19 December 2013, Singapore*, pages 667–684, 2013.

[JXYY10] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:221–231, 2010.

[KB91] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 1398–1404 vol.2, April 1991.

[KBR12] Scott Kiesel, Ethan Burns, and Wheeler Ruml. Abstraction-guided sampling for motion planning. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*, 2012.

[KF11] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal

motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[KFT+08] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How. Motion planning for urban driving using rrt. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1681–1686, Sept 2008.

[KG15] Kristofer D. Kusano and Hampton C. Gabler. Comparison of expected crash and injury reduction from production forward collision and lane departure warning systems. *Traffic Injury Prevention*, 16(sup2):S109–S114, 2015. PMID: 26436219.

[KH89] Y. Kanayama and B. I. Hartman. Smooth local path planning for autonomous vehicles. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 1265–1270 vol.3, May 1989.

[Kha85] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, March 1985.

[KHB09] Matthias Jürgen Kühn, Thomas Hummel, and Jenoe Bende. Benefit estimation of advanced driver assistance systems for cars derived from real-life accidents. 2009.

[KKY+16] G. Kang, Y. B. Kim, W. S. You, Y. H. Lee, H. S. Oh, H. Moon, and H. R. Choi. Sampling-based path planning with goal oriented sampling. In *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1285–1290, July 2016.

[KL00] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, April 2000.

[KL02] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 968–975 vol.1, May 2002.

[KLF05] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning a*, 2005.

[KLY14] D. Kim, J. Lee, and S. Yoon. Cloud rrt*: Sampling cloud based rrt*. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2519–2526, May 2014.

[KSL+19] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin. Probabilistic completeness of rrt for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[KTF+09] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, Sept 2009.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. cite arxiv:1312.6114.

[KWP+11] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the rrt*. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483, May 2011.

[Lav98] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[LBM+18] Konstantinos Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis D. Bochtis. Machine learning in agriculture: A review. In *Sensors*, 2018.

[LCLX18] Y. Li, R. Cui, Z. Li, and D. Xu. Neural network approximation based near-optimal motion planning with kinodynamic constraints using rrt. *IEEE Transactions on Industrial Electronics*, 65(11):8718–8729, Nov 2018.

[LFV+16] Colin Lea, Michael D. Flynn, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks for action segmentation and detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1003–1012, 2016.

[LJJK01] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[LL03] Stephen R. Lindemann and Steven M. LaValle. Current issues in sampling-based motion planning. In *ISRR*, 2003.

[LRWW98] Jeffrey Lagarias, James Reeds, Margaret Wright, and Paul Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9:112–147, 12 1998.

[MBS06a] K. Macek, M. Becker, and R. Siegwart. Motion planning for car-like vehicles in dynamic urban scenarios. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4375–4380, Oct 2006.

[MBS06b] K. Macek, M. Becker, and R. Siegwart. Motion planning for car-like vehicles in dynamic urban scenarios. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4375–4380, 2006.

[MLB+18] M. Müller, X. Longl, M. Betsch, D. Böhmländer, and W. Utschick. Real-time crash severity estimation with machine learning and 2d mass-spring-damper model. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2036–2043, Nov 2018.

[MW04] D. S. Meek and D. J. Walton. An arc spline approximation to a clothoid. *J. Comput. Appl. Math.*, 170(1):59–77, September 2004.

[NBS17] Parthasarathy Nadarajan, Michael Botsch, and Sebastian Sardina. Predicted-occupancy grids for vehicle safety applications based on autoencoders and the random forest algorithm. pages 1244–1251, 05 2017.

[NMC10] A. A. Neto, D. G. Macharet, and M. F. M. Campos. Feasible rrt-based path planning using seventh order bézier curves. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1445–1450, Oct 2010.

[ODZ+16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[OF16] Michael Otte and Emilio Frazzoli. Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research*, 35(7):797–822, 2016.

[PA15] L. Palmieri and K. O. Arras. Distance metric learning for rrt-based motion planning with constant-time inference. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 637–643, May 2015.

[Pac06] H.B. Pacejka. *Tyre and Vehicle Dynamics*. Automotive engineering. Butterworth-Heinemann, 2006.

[PC14] Pedro O. Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–82–I–90. JMLR.org, 2014.

[PCY+16] Brian Paden, Michal Cáp, Sze Zheng Yong, Dmitry S. Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1:33–55, 2016.

[PHCM18] N. Pérez-Higueras, F. Caballero, and L. Merino. Learning human-aware path planning with fully convolutional networks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–5, May 2018.

[Pol64] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17, 1964.

[PZP17] Abhinav Podili, Chi Zhang, and Viktor K. Prasanna. Fast and efficient implementation of convolutional neural networks on fpga. *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 11–18, 2017.

[QY18] Ahmed H. Qureshi and Michael C. Yip. Deeply informed neural sampling for robot motion planning. *CoRR*, abs/1809.10252, 2018.

[RHN16] Pongsathorn Raksincharoensak, Takahiro Hasegawa, and Masao Nagai. Motion planning and control of autonomous driving intelligence system based on risk potential optimization framework. *International Journal of Automotive Engineering*, 7(AVEC14):53–60, 2016.

[SD13] Disha Sharma and Sanjay Kumar Dubey. Anytime a * algorithm – an extension to a * algorithm. 2013.

[SDSS09] E. Shan, B. Dai, J. Song, and Z. Sun. A dynamic rrt path planning algorithm based on b-spline. In *2009 Second International Symposium on Computational Intelligence and Design*, volume 2, pages 25–29, Dec 2009.

[SHM+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[SMW17] K. Shiarlis, J. Messias, and S. Whiteson. Rapidly exploring learning trees. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1541–1548, May 2017.

[SPK11] P. Szulczyński, D. Pazderski, and K. Kozłowski. Real-time obstacle avoidance using harmonic potential functions. *Journal of Automation Mobile Robotics and Intelligent Systems*, Vol. 5, No. 3:59–66, 2011.

[SR16] Rajiv Shah and Rob Romijnders. Applying deep learning to basketball trajectories. *CoRR*, abs/1608.03793, 2016.

[SRW08] Robin Schubert, Eric Richter, and Gerd Wanielik. Comparison and evaluation of advanced motion models for vehicle tracking. *2008 11th International Conference on Information Fusion*, pages 1–6, 2008.

[Ste94] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3310–3317 vol.4, May 1994.

[T.06] Mitchell T. The discipline of machine learning. , School of Computer Science, Carnegie Mellon University, Pittsburg, USA, 2006.

[US03] C. Urmson and R. Simmons. Approaches for heuristically biasing rrt growth. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots*

and Systems (IROS 2003) (Cat. No.03CH37453), volume 2, pages 1178–1183 vol.2, Oct 2003.

[Var93] P. Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, Feb 1993.

[VF04] Dizan Vasquez and Thierry Fraichard. Motion prediction for moving objects: a statistical approach. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, 4:3931–3936 Vol.4, 2004.

[Vid97] M. Vidyasagar. *A Theory of Learning and Generalization: With Applications to Neural Networks and Control Systems*. Springer-Verlag, Berlin, Heidelberg, 1997.

[Von18] V. Vonásek. Motion planning of 3d objects using rapidly exploring random tree guided by approximate solutions. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 713–720, Sept 2018.

[WAS99] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. Maprm: a probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1024–1031 vol.2, May 1999.

[Wes] Justin T. Westbrook. Motorcyclist sues gm over crash with self-driving chevy bolt. [Online; accessed August 19, 2019].

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.

[WM03] S. Waydo and R. M. Murray. Vehicle motion planning using stream functions. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 2484–2491 vol.2, Sep. 2003.

[WWIT16] Thorsten Wuest, Daniel Weimer, Christopher Irgens, and Klaus-Dieter Thoben. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45, 2016.

[WXL+10] Wei Wang, Xin Xu, Yan Li, Jinze Song, and Hangen He. Triple rrts: An effective method for path planning in narrow passages. *Advanced Robotics*, 24(7):943–962, 2010.

[Yan13] K. Yang. An efficient spline-based rrt path planner for non-holonomic robots in cluttered environments. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 288–297, May 2013.

[YGS13] Kwangjin Yang, Seng Keat Gan, and Salah Sukkarieh. A gaussian process-based rrt planner for the exploration of an unknown and cluttered environment with a uav. *Advanced Robotics*, 27(6):431–443, 2013.

[YS10] K. Yang and S. Sukkarieh. An analytical continuous-curvature path-smoothing algorithm. *IEEE Transactions on Robotics*, 26(3):561–568, June 2010.

[ZKB07] Matthew Zucker, James Kuffner, and Michael Branicky. Multipartite rrts for rapid replanning in dynamic environments. In *Proc. IEEE Int. Conf. Robotics and Automation*, April 2007.

[ZWWW09] Q. Zhu, Y. Wu, G. Wu, and X. Wang. An improved anytime rrts algorithm. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 1, pages 268–272, Nov 2009.