UNIVERSITY OF MISKOLC

Intelligent Solutions for Autonomous Vehicle Driving System Optimization

by

Ahmad REDA

Supervised by Dr. József VÁSÁREHELYI

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
József Hatvany Doctoral School for Computer Science and Engineering
Faculty of Mechanical Engineering and Informatics
Institute of Automation and Infocommunication

August 2023

Declaration of Authorship

I, Ahmad REDA, declare that this thesis titled, 'Intelligent Solutions for Autonomous Vehicle Driving System Optimization' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____

Date: ____

UNIVERSITY OF MISKOLC

Abstract

Faculty of Mechanical Engineering and Informatics
Institute of Automation and Infocommunication

Doctor of Philosophy

by Ahmad REDA

Developing autonomous vehicles is a highly important topic in automotive industry and the field of intelligent transportation systems. A variety of classical control strategies have already proved their merits in this field. However, with the increase in the non-linearity and complexity of the driving system's environment, the efficiency of these approaches drop off due to the limitations of their computing capabilities with such highly complex systems or to lack of efficacy related to maintaining the balance between driving performance and driving smoothness. For example, model predictive control (MPC) is very well known classical control strategy that is used for steering system due to its abilities in solving the optimization problem in real time, handling the system constraints and dealing with changing dynamics of the vehicle. However, its efficiency is negatively affected in high complex environment and may not be able to meet the real time requirements due to the fact that it solves the optimization problem at each time step which massively increases the computational loads. As a result, developing robust systems becoming more crucial and remains an open challenge for researchers and automotive companies alike. The motivation in this work was to contribute to the optimization of the autonomous vehicle driving system. we tackled this problem in two different aspects. In the first part, we focused on optimizing the the implementations of the classical control, precisely MPC control on limited resources platform (low-end FPGA). It is certainly noticeable that the use of artificial intelligence (AI) in this field is unavoidable due to the efficiency that has been achieved in different fields. In the second part we focused on taking advantage of machine learning algorithms to provide an efficient alternative solutions to the classical control. In addition to optimize the deployment of DNN on FPGA using a new innovative tool.

Acknowledgements

I am deeply indebted to my esteemed supervisor, Dr. József Vásárhely, for his exceptional mentorship, support, and invaluable guidance that have profoundly shaped my doctoral journey. Dr. Vásárhely's unwavering commitment to academic excellence, coupled with his open-door policy, has fostered an environment where I felt encouraged to explore new ideas, seek guidance, and engage in thought-provoking discussions. His expert knowledge, and friendly demeanor have not only elevated the quality of my research but also enriched my overall academic experience, leaving an indelible mark on my intellectual growth. I also, would like to thank Dr. Ahmed Bouzid for his support and invaluable contributions. In addition, I would like to express my gratitude to Dr. Attila Trohák and my friends from the Institute of Automation and Info-communication. Their support, understanding, and encouragement have been instrumental in my academic pursuits. Last, but certainly not least, I can't stop thanking Dr. Ildi Bölkény, my family and all friends.

This research was supported by the European Union and the Hungarian State, co-financed by the European Regional Development Fund in the framework of the GINOP-2.3.4-15-2016-00004 project, aimed to promote the cooperation between the higher education and the industry.

Contents

List of Figures

List of Tables

Abbreviations

DOF	Degrees of Freedom
MRAC	Model Reference Adaptive Controller
MPC	Model Predictive Control
ASIC	Application-Specific Integrated Circuits
GPU	Graphics Processing Units
FPGA	Field Programmable Gate Arrays
AI	Artificial Intelligence
PS	Processor System
PL	Programmable Logic
AXI	Advanced eXtensible Interface
IP	Intellectual Property
ADC	Analog to Digital Converter
FF	Flip-Flops
CLB	Customizable Logic Block
LUT	Lookup Table
LiDAR	Light Detection and Ranging
CNN	Convolution Neural Network
DNN	Deep Neural Network
DPU	Data Process Unit)
HDL	Hardware Description Language
HLS	High Level Synthesis
TSR	Traffic Sign Recognition
XSG	Xilinx System Generator
DSP	Digital Signal Processing
SoC	System on Chip

UAM	Urban Air Mobility
FLS	Fuzzy Logic System
QoR	Quality of Result
SVD	Singular-Value Decomposition
HIL	Hardware In Loop
HW	Hardware
HwCoSim	Hardware Co-Simulation
LSTM	Long short-term memory
MIMO	multi-input multi-output
QP	Quadratic Programming
LP	Linear Programming
HEV	Hybrid Electric Vehicles
IoT	Internet of Things
LTI	Linear Time Invariant
LPV	Linear Parameter-Varying
KF	Kalman Filter
AMPC	Adaptive Model Predictive Controller
QP	Quadratic Problem
RMSE	Root Mean Square Error
FPAA	Field Programmable Analog Arrays
SGD	Stochastic Gradient Descent
ADAM	Adaptive Moment Estimation
RMSProp	Root Mean Square Propagation
DQN	Deep Q-learning
RL	Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
GWO	GreyWolf Optimizer
MDP	Markov Decision Processes
SGD	Stochastic Gradient Descent
PG	Policy Gradients
DOG	Deterministic Policy Gradients
ADAS	Advanced Driver Assistance Systems
ACC	Adaptive Cruise Control

LTL	Linear Temporal Logic
CBF	Control Barrier Function
CACC	Cooperative Adaptive Cruise Control
PI	Proportional Integral
MO	Measured Outputs
MD	Measured Disturbance
MV	Manipulated Variable
ReLU	Rectified Linear Unit

Chapter 1

Introduction

Autonomous vehicles have been researched since 1980, for the time, these researches represented impressive technological advancements. In the 1980s, the Defense Advanced Research Projects Agency put its first 600-meter-distance prototypes on the road. In 2004, the same agency introduced the "DARPA Challenge," which encouraged institutions to innovate in this area. The objective was traveling 240 kilometers through the Mojave Desert without human assistance. Over time, more innovations have gained popularity, and diverse companies and research centers have taken up the challenge of developing a fully autonomous vehicle. Tesla, Waymo, Zoox and automakers like Mercedes and Ford are currently the most well-known in this field [BGC+21], [TMD+06].

The automotive industry has agreed on a definition of autonomous driving, and it is best summarized as" the ability of the vehicle to drive partly or fully without or with limited human interaction". According to the Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems "SAE-J3016", vehicle autonomy is divided into six different levels, from fully manual (level 0) to fully autonomous (level 5). Level 0 (No Automation) depends on the human driver to perform all the driving tasks, it is manually controlled. Level 1 (Driver Assistance) is considered as the lowest automation level, where the driver has full responsibility, but some assistant driving systems are included for certain circumstances. Level 2 (Partial Automation) combines different automated functions which can be working simultaneously, such as steering and acceleration tasks, but the driver is still involved in the driving tasks such as performing the maneuvers and has to monitor the environment all the time. At Level 3 (Conditional Automation) the vehicle has the capability of detecting the surrounding environment and making decisions in normal conditions, but the necessity of the driver still exists, meaning that the driver has to be ready to take control over the vehicle at any time. At Level 4 (High Automation) the vehicle performs all the driving tasks in most circumstances, and the driver still has the option to take control. At Level 5 (Full Automation) the vehicle is capable of performing all driving tasks in all circumstances, and the driver has the option to manually override [Com14], [UDoT18]. Generally speaking, the vehicle needs to be able to coordinate and effectively implement functions under three main pillars in order for it to be able to partially or fully drive. Observing the driving surroundings to detect risks and emergency scenarios, then automatically taking action to protect the passengers and eliminate potential collisions. The Advanced Driving Assistance Systems (ADAS), which include functions like driving assistance, collision protection, and emergency breaking keeps track of safety-related issues. The ADAS technologies are anticipated to advance and play a crucial role in the autonomous driving system optimization. Autonomous vehicle are made up of three main parts, the vehicle, driving software, and hardware, where it depends on sensor, actuators, complex control algorithms, and powerful processors in order to perform its tasks [DTP21]. The core functions of the autonomous vehicle can be categorised into three main categories: perception, planning, and control as figure 1.1 shows. The environment perception provides the vehicle with the required information about the surrounding driving environment, including the vehicle's location, the drivable areas, the velocity, etc. Different sensors and tools can be implemented to tackle the perception task, such as using ultrasonic sensors, cameras, LiDARs (Light Detection And Ranging), or even a combination of these (sensor fusion) to decrease the uncertainty of the data. Based on the collected data, the best scenarios are obtained and the required control actions are made in the planning module in order to drive the vehicle efficiently to the desired location. In the control function, the commands are sent to the actuators to put the control strategy into action

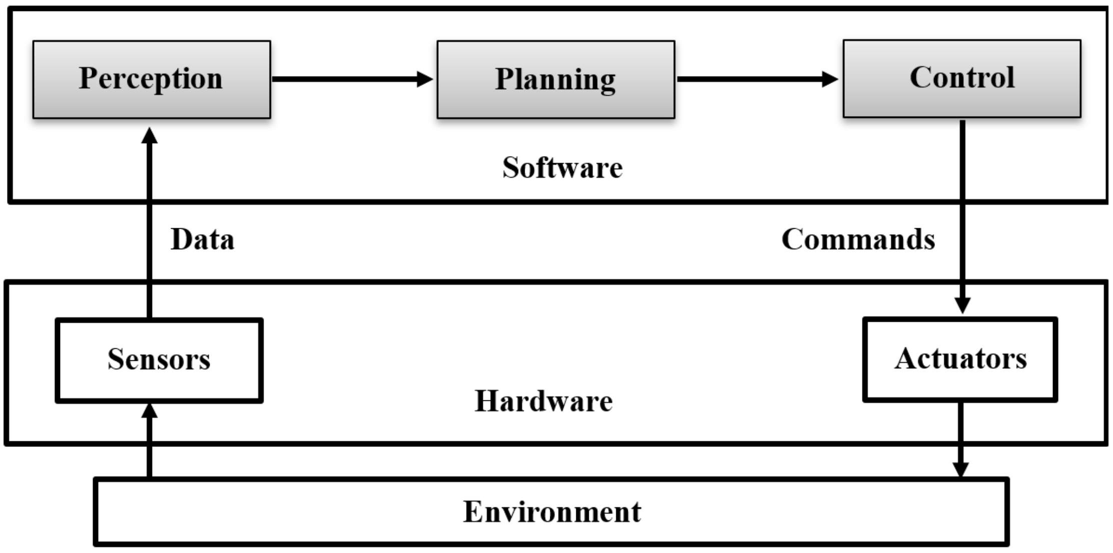

FIGURE 1.1: The interaction of the autonomous vehicle within the surrounding environment.

[BSDD17]. Adaptive behaviour in autonomous vehicles provide the ability to changes their behavior parameters in accordance with their environment, while a the autopilot technology is used to automatically manage a vehicle's operation without any manual control. Essentially, the core of autonomous driving is the development of systems that can automate the function of driving. Despite the fact that autonomous vehicles have been around for a while, the demand to develop one that is completely functional has accelerated in the past decade.

In a scientific point of view, the work presented in this document globally proposes a research, in the context of automated driving and precisely a contribution in automated steering and safety. After the introduction where an overview of the autonomous vehicles is presented, the necessary theoretical and hardware background of the different aspects of the researches including control strategies for path tracking, Re-configurable Computing and Hardware Acceleration, and the recent machine learning-based applications in the field are presented in the second and the third chapters. Then comes to the contribution part which is provided in the next 4 chapters. The contributions in chapter 4 can be summarised in two main points. First, studying, analysing and improving the implementations of the MPC controller for the task of automated driving especially with changing dynamic systems. Second, the use of rapid prototyping method (hardware/-software co-design) to deploy the design on FPGA (hardware-embedded system). The suggested solution in chapter 5 is to develop a deep neural network model based on the behavior of the traditional MPC controller so that the DNN model can replace the MPC controller in high complexity driving system environments. Additionally, one of the motivations behind the work is to propose an alternative tool to implement deep neural networks on low-end FPGA. The main contributions in chapter 6 can be summarized in two main points. The first is leveraging the advantages of reinforcement learning and supervised learning by combining them in one control model in such a way that the reinforcement learning-based model optimizes the actions that are taken by the supervised neural work (DNN) model. The second contribution comes in enriching the research on RL algorithms and paving the way to bring RL closer to real-world implementations. The contribution of chapter 7 comes in the orientation of enriching the studies that have been conducted on reinforcement learning method in the frame of safety automated driving in order to validate its efficiency and stability compared to the used classical control approaches. The last chapter summarise the entire work and the provided theses.

Chapter 2

Background

This chapter presents a scientific background for the major topics covered in the performed works including the commonly used vehicle models and control strategies for path tracking task of autonomous vehicles. In addition to the Re-configurable Computing and Hardware Acceleration.

2.1 Autonomous vehicles model and control strategy for path tracking tasks

Generally speaking, building the vehicle model is a crucial stage in the development process. Vehicle (like a robot) model can be divided into two main classes, holonomic system, and non-holonomic system based on the number of controllable degrees f freedom (DOF) against the total number of DOF. In a holonomic system, the number of controllable degrees of freedom is equal to the total number of DOF. In contrast, a system with a number of controllable degrees of freedom less than the total number of DOF is called a non-holonomic system. It is worth mentioning that most of the vehicles are considered non-holonomic due to the fact that only two degrees of freedom are controllable (the lateral positions and the longitudinal directions) [CQCD15], [RNS04]. One more thing to take into consideration is that the vehicle model can be reduced to two tire model which is called a bicycle model. In bicycle models, the right and the left tires are assumed to have the same behavior. The autonomous vehicle's motion in regard to tracking tasks can be divided into three main types, point-to-point motion, trajectory tracking, and path following. Point-to-point motion is the basic task, where the goal for the vehicle is to reach the desired point starting from the initial one with no regard for a specific path. The goal of the vehicle in the case of a path-following task is to reach the desired point starting from an initial one by driving the car through a specific geometric path with no regard for time constraints. In case of the trajectory tracking, the goal is to drive the vehicle from the initial point to the desired one based on a specific geometric path with a time concern. In this section, the most commonly used vehicle models and control strategies for path-tracking tasks are described.

2.1.1 Geometric Path Tracking

Due to its simplicity, geometric path tracking is considered one of the most commonly used autonomous vehicle models for tracking tasks. In the geometric vehicle model, only the dimensions and the positions of the vehicle are taken into consideration with no regard for internal or external forces, velocity, or acceleration . The geometric model was developed based on the Ackerman Configuration as it is described in Figure 2.1, and the steering angle of the vehicle is defined by equation 2.1, where L is the distance between the axles of the front and the rear wheels and R is the radius of the turn [Sni09].

Several controllers for autonomous vehicle steering control were provided based on the geometric model. Geometric controllers are considered the most common and simple type of controllers in the field of path tracking due to their stability and simplicity, where the state variables are simple with the absence of derivatives. The common control strategies used for this model can be divided into three main categories which are, Follow the Carrot, Pure Pursuit, and Stanley Method. The idea behind the "Follow the Carrot" strategy is very simple, where a point on the desired path is chosen to be the current destination of the vehicle. The chosen point is called the carrot point and it should be chosen at one look-ahead distance from the vehicle. In order to drive the car toward the carrot point, the steering angle should be determined to minimize the orientation error which is the difference between the orientations of the vehicle and the carrot point. The steering angle δ is calculated as equation 2.2 shows where θ_e: is the difference between the vehicle orientation and the carrot point orientation, while k_{carrot} is the proportional controller used to reduce the error. Briefly, the vehicle chases a moving point through the desired bath. The look-ahead distance is a crucial parameter and should be determined accurately. If the distance is too large, the vehicle will be driven to the carrot point with no consideration for the corners of the path. Conversely, if the distance is too small, the vehicle will oscillate especially at high speed. In order to minimize the oscillation problem and achieve smooth driving, "Pure pursuit" which is very similar to the "Follow the Carrot" method, uses another technique to determine the steering angle in order to drive the car toward the goal point. The steering angle is determined in relation to a circular arc which is a line connected and goes through the current position of the vehicle and the goal (carrot) point as equation 2.3 shows [Sni09], [AZHK17], where L_d is look-ahead distance, L is the dimension of the vehicle, and L_e is a lateral error. The steering angle in the Stanley method is calculated based on two main factors, lateral error and heading error as equation 2.4 shows, where v is the vehicle velocity and k is a gain parameter [HTMT07], [TMD+06]. It is worth pointing out that the method name "Stanley" is the name of an autonomous vehicle that was developed by the Stanford University team during the participation in the second DARPA Grand Challenge in which the team achieved first place. Figure 2.2 shows the main parameters used in the geometric controllers.

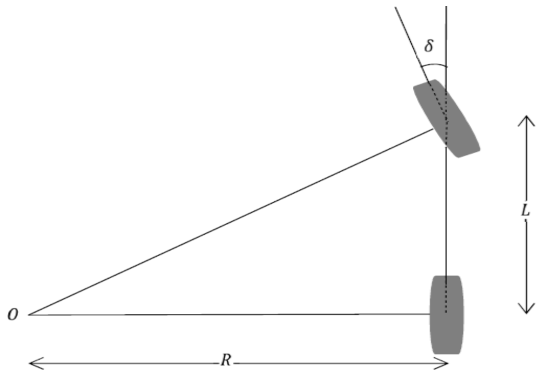

FIGURE 2.1: The Geometric Model (Ackerman configuration).

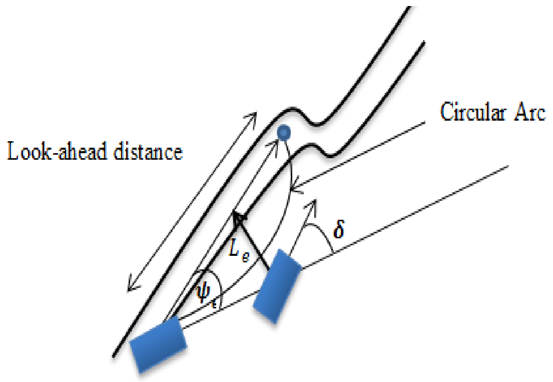

FIGURE 2.2: Configuration of Geometric Controllers.

$$\delta = tan^{-1}\frac{L}{R} \tag{2.1}$$

$$\delta = K_{carrot}\theta_e \tag{2.2}$$

$$\delta = tan^{-1}\frac{2LL_e}{L_d} \tag{2.3}$$

$$\delta = tan^{-1}\frac{kL_e}{v} \tag{2.4}$$

2.1.2 kinematic Path Tracking

The kinematic term in general, concerns studying the motions of the body with no regard for the internal or external forces. In this section, the vehicle model and the control strategies for kinematic path tracking are described.

Unlike the geometric vehicle model, the kinematic model describes the motion of the vehicle taking into consideration the velocity and the acceleration with no regard for its internal forces (internal dynamics). The kinematic model is usually used to study the lateral position and yaw angle. As mentioned earlier, the full vehicle model can be reduced to the bicycle model. In this case, the direction of the longitudinal vehicle's direction is the same as the vehicle heading direction. Figure 2.3 and Equation 2.5 describe the kinematic vehicle model (bicycle model), where v_x and v_y are the velocity in the local coordinates (X-Y), v_X and v_Y are the velocity in the global coordinates (X-Y), δ is the steering angle and θ is the vehicle orientation, v is the total velocity and ω is yaw rate of the vehicle or $\hat{\theta}$ [RGNR+09], [JSS09]. (kinematic model).

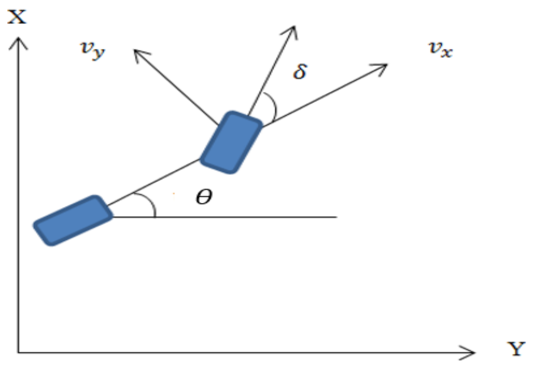

FIGURE 2.3: Kinematic Vehicle Model.

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \hat{\theta} \end{bmatrix} \tag{2.5}$$

The development of the kinematic control strategy is mostly based on the kinematic vehicle model integrated with the geometric model without taking into consideration the vehicle's dynamic. Several interesting studies provided in regard to kinematic controlling. Sun et [ZCN+12] presented a study to address the problem of path tracking for the autonomous vehicle and analyses the relationship between the road model and path tracking method. New methods were provided called "Ribbon Model" for vehicle–road and "Ribbon Tracking Method" for path tracking. De Luca et al [AOS05] provided a comparison study of different feedback solutions for different tasks such as path tracking and stabilization for car-like robots.

2.1.3 Dynamic Path Tracking

Kinematic and Geometric models are widely used and effective for systems where there is no need to take internal and external forces into consideration. On the other hand, these forces should be taken into consideration under specific conditions such as a sharp trajectory curvature. Ignoring the vehicle dynamic for such conditions negatively affects the performance and the safety aspect In the dynamic vehicle model, the motion of the vehicle is described with respect to its position, velocity, and acceleration, taking into consideration the applied internal and external forces such as the gravity force (G) and the lateral forces (F_f, F_r) as it is shown in Figure 2.4, which also describes the parameters that are taking into consideration in the designing process of the vehicle model, where (θ) is the vehicle heading,(δ_f, δ_r) are steering angles of the wheels, (L) is the vehicle's wheelbase, (L_f, L_r) are the half wheelbase and (β_f, β_r) are the side slip angles of the wheels, noting that r and f refer to rear and front wheels respectively.

In the path-tracking task of autonomous vehicles, the control law of the dynamic controller includes the dynamic properties of the system. Taking the effects of the vehicle dynamics into consideration naturally makes the dynamic controllers more efficient and stable compared to geometric and kinematic controllers [NHZHAK11]. However, dynamic feedback (such as the torque) is required for these control strategies, which in turn requires a special type of sensor and more data processing. Consequently, dynamic controllers are more expensive in terms of cost and computational loads [[PCvL11], [NSMN10].

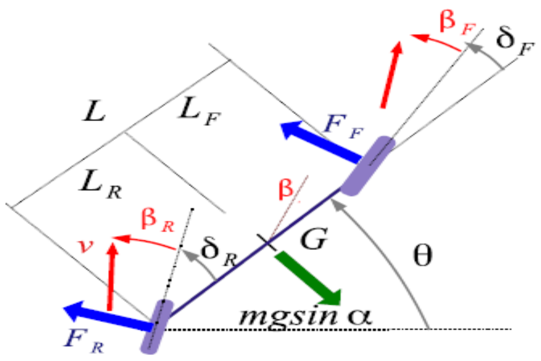

FIGURE 2.4: Dynamic vehicle model [DLTR17]

2.1.4 Adaptive and Model Predictive Control

The adaptive controller is developed to deal with systems that have uncertain, unknown, or changeable parameters. Meaning that the main objective of using the adaptive controller is to provide a robust control system under uncertain and changeable environments. The plant parameters are estimated and used in the model in order to adjust the controller [ch212]. Adaptive controllers are widely used for autonomous vehicle tasks. Martins et al [FNCMBF08] used an adaptive controller for vehicle path-tracking tasks and the proposed model used linear and angular velocity as a reference signal. The proposed controller provided a stable control system with high performance. Dørum et al [JUG15] provided a comparison between two adaptive controllers for vehicle trajectory tracking tasks. For the first controller, the adaptive law was implemented to estimate the unknown parameters of the model, while in the case of the second controller, a Model Reference Adaptive Controller (MRAC) was provided. The compression shows that the MRAC achieved the best performance. Machine Learning is widely used with adaptive controllers in order to improve control decisions in terms of speed and accuracy. In the study [XHL$^+$18], a lateral motion control method was provided where the objective of the suggested method is to maintain the yaw stability and minimize the tracking error. Model Predictive Control (MPC) is a multi-variable feedback control strategy that solves an online optimization problem, taking into consideration the interactions between the variables of the target system. Based on the plant model, the MPC controller calculates its inputs and uses an optimizer to ensure that the output of the plant follows the reference output. MPC uses predicts the outputs based on the future prediction strategy. MPC controller simulates several future scenarios and the optimizer chooses the best scenario based on the cost function which represents the error between the reference target and the predicted outputs, where the optimal scenario corresponds to the minimum cost function. Figure 2.5 shows the main structure of the model predictive control strategy [PBT$^+$08].

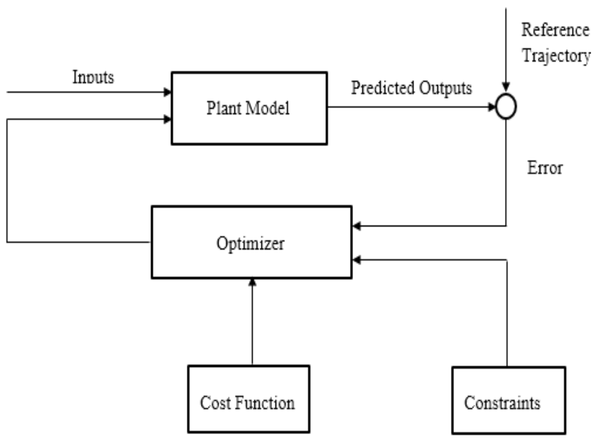

FIGURE 2.5: Model Predictive Controller.

2.2 Re-configurable Computing and Hardware Acceleration

The computing architecture research community is currently dealing with the explosive scale of diverse data explosion due to the rising big data applications like Machine

Learning, Voice Recognition, and DNA Sequencing in recent years. Therefore, the necessity of exploiting the parallelism and acceleration provided by hardware is increased and the studies are inducted to make data-intensive computing highly efficient. In this context, heterogeneous accelerators have become a popular topic in the computer architecture field. The deployment of heterogeneous accelerators currently primarily relies on heterogeneous computing components including application-specific integrated circuits (ASIC), graphics processing units (GPU), and field programmable gate arrays (FPGA). ASIC-Application Specific Integrated circuit – designed by the user and produced by chip maker, optimized from the point of view of size, computation speed and power consumption. However, the long design cycle, the high cost, and the lack of flexibility are the main disadvantages of ASIC. On the other hand, FPGA is a type of reconfigurable device, that combines the adaptability of software with the effectiveness of hardware computing, resulting in a design cost for FPGA-based hardware accelerators that is significantly cheaper than that of ASIC- ones. It is made up of hardware blocks with user-programmable interconnects to modify the functionality of a specific application, with good customization and scalability. This flexibility makes FPGA a suitable choice for applications where the standards are changeable. Overall, accelerator based on reconfigurable hardware has advanced significantly in recent years, not only in terms of hardware but also in terms of software and algorithms. Improve the dependability of reconfigurable architecture, which offers a strong foundation for creating a new type of computer system and encouraging the industrialization of AI chips and systems. In comparison to CPU and GPU, FPGA is essentially an instructionless, shared-memory-less chip that more effectively speeds algorithms, which encompasses numerous compute-intensive and communications-intensive tasks. Over the past decade, numerous well-known companies, like Google and Intel, as well as academic institutions have demonstrated a keen interest in this field. In 2014, The HARP project was proposed by Intel after acquiring FPGA producer Altera. This project aimed to develop a teaching-related platform [CLG+17]. At the same year, IBM successively released and makes an FPGA Accelerator service available for the developers in order to provide them with a platform for creating, designing, and testing in many emerging areas [JGCC15]. The F10A, a jointly designed FPGA accelerator card by Inspur and Intel, was released in November 2016. which fulfills the high-density, and high-performance demands of Open-CL. The powerful Adaptive Compute Acceleration Platform (ACAP) has been introduced after AMD acquired Xilinx. ACAP is a fully software-programmable, heterogeneous compute platform that combines Scalar Engines, Adaptable Engines, and Intelligent Engines [XIL20]. Many studies on heterogeneous reconfigurable systems are also carried out in the academic field such as Xilinx's Zynq-7000 [CEES14], ZCluster [ZC13] at the University of Toronto, and others. The Zynq-7000 [CEES14], a programmable heterogeneous multi-core system-on-chip introduced by Xilinx Inc. It has great performance, flexibility, and reconfigurability and is made up of programmable logic devices and processing systems. ARM processor serves as the processing system's core, which can function independently.

Prior to the development of the Zynq, processors were coupled with the FPGA which complicated the communication between the Processor System (PS) and Programmable Logic (PL). With the Zynq architecture, a dual-core ARM Cortex-A9 processor is combined with a conventional (FPGA) where Advanced extensible Interface (AXI) serves as the interface between the different components of Zynq architecture. AXI enables communication with low latency and high bandwidth. Users were using a soft core processor like Xilinx's Microblaze before integrating the ARM CPU within the Zynq device. The adaptability of the processor instances inside the design is the primary benefit of Microblaze. Figure 2.6 presents Zynq overall view.

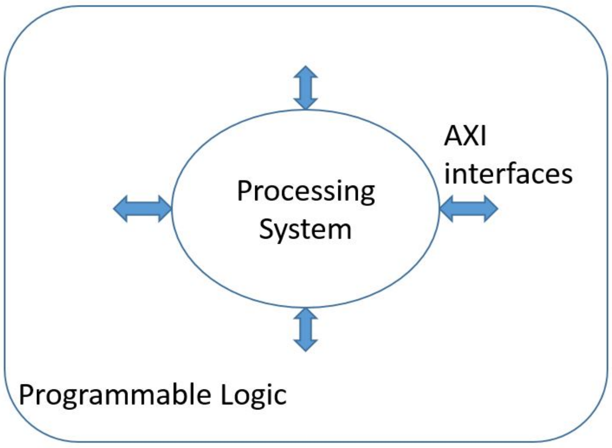

FIGURE 2.6: Overall view of Zynq architecture.

Generally speaking, the design flow of Zynq architecture goes through several steps. In the first step, the specifications and requirements of the system are to be defined. In the second step, the different functions are to be assigned in either the processor system or programmable logic. Developing and testing hardware and software come next. In terms of the PL part, the tasks are to determine the function blocks that are required to meet the design characteristics, create the IPs (Intellectual Property) and connect them. For the software part, the task is to develop code to run on the PC. High-level languages like C++ have made programming comparatively simple compared to thousands of assembly lines of code. High-level languages, on the other hand, also seem to be a dead end with a highly complex design which rises the necessity for more efficient tools. The VIVADO IP integrator, a graphic tool for automatic HDL code generation, is utilized in the case of Xilinx systems designs. As a result, system integration and testing are necessary to complete the design. Figure 2.7 presents the architecture of the Xilinx Zynq family. It shows that the processor communicates with flash memory controllers, SDRAM, on-chip memory, and peripheral blocks using ARM AMBA AXI-based interface. The Processing System is made up of these parts collectively. Two CAN controllers are among the common interfaces that the PS fixed peripheral control block offers. Analog-to-digital converters (ADCs), which are also included in Zynq devices, allow for the direct connection of external sensors without the requirement for external digitization. Using multiple ARM AMBA AXI connections coupling between the two crucial elements of the Zynq architecture and PL, providing a high bandwidth that enables a high-speed data transfer. The Zynq SoC's programmable logic component, like other FPGAs, is made up of customizable logic blocks (CLBs) that comprise slices. Flip-flops (FFs), switch Matrix, and look-up tables (LUTs) are all present in each slice, in addition to Block RAMs and DSPs.

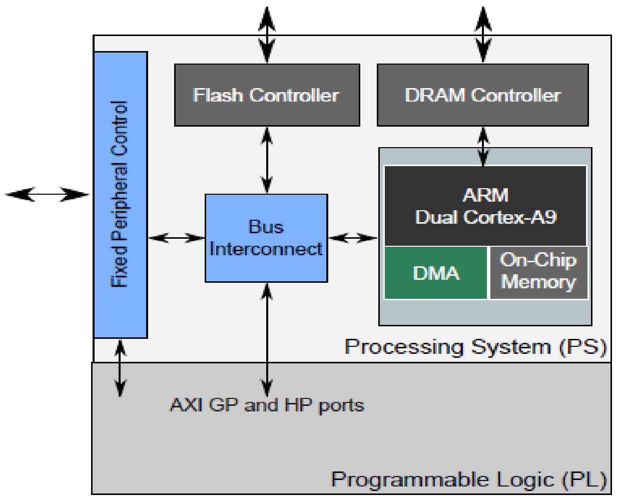

FIGURE 2.7: Zynq architecture-Processor Subsystem and Programmable Logic [VSFA14]

2.3 Chapter Summary

In this chapter, I introduced the main topics covered in this work. An overview of the vehicle models and control strategies for path tracking tasks and the Re-configurable Computing and Hardware Acceleration which is related to the hardware used to perform the implementations of the suggested algorithms. These concepts pave the way for a better understanding of the next chapters.

Chapter 3

Machine learning and FPGA -Applications and Methods

This chapter focuses on providing a detailed academic review of the latest control methods including machine learning algorithms to perform different autonomous vehicle tasks. This chapter also focuses on reconfigurable computing accelerators in both industry and academia, mainly covering FPGA-based accelerators and application-specific solutions, presenting the achieved results and the implementation concerns and challenges.

3.1 Introduction

Machine Learning is a power full method used to build complex models to perform a prediction task using data. It becomes one of the most interesting research areas in the last dedicates. It provides effective solutions for many complex tasks and a wide range of applications [PSN+17], [LYS18], [GDDM14a]. Machine Learning can be categorized into three main categories, supervised learning, Unsupervised learning, and Reinforcement learning. In terms of embedded systems, the applications of Machine Learning take advantage of existing developed hardware such as FPGA which is known for its high computational capability and the limitation constraints in power consumption [AMM+18]. Machine learning accelerators have the potential to greatly improve performance. ML has been applied in a variety of applications, especially in complex and multi-task applications such as autonomous vehicles [BMKY18]. I address the problems and presented the solutions in both the software and hardware space. Autonomous vehicles are equipped with multiple sensors such as cameras, radars, and LiDARs, to perceive the surrounding environment. These sensors generate huge data, hence real-time processing of this data requires high-performance computing systems. Instead of using CPUs and GPUs for implementing AI, FPGAs are adopted since they feature high performance with low power consumption. This chapter presents the latest solutions for autonomous vehicles involving AI implemented on FPGAs.

3.2 Autonomous vehicle applications-specific solutions

Driving an autonomous vehicle within an uncertain environment is a very critical task, therefore, the necessity of developing efficient algorithms to overcome this challenge is increased. The researchers in [DTMD08] suggested a method to deal with the complexity of creating a path planner and performing the optimization for continuous variables in free areas. This method mainly depends on a "Search Algorithm", which consists of two main steps. In the first step, the sub-paths which is away from the goal are excluded based on the fact that this heuristic can be processed using of-line data. After that, it can be formulated to correspond to the current goal. The second step is used to determine the U shaped and the dead ending in order to determine the shortest path to the goal by using the obstacle map. The proposed method uses a heuristic search by applying the Hybrid A^* algorithm. However, there are different types of search algorithms including A^* and D^*. Figure 3.1 shows the main differences between the three methods. Left: A^* only moves to the states that correspond to the grid-cell centers and the cost function is associated with the center of the cells. Center: D^*, the cost function is associated with the cell corners, and arbitrary linear pathways between cells are permitted. Right: Hybrid A^*, where each cell is associated with a continuous state, and the cost of that continuous state determines the cell's cost. The hybrid A* provides a sub-optimal path plan which needs to be improved. In this paper, the researchers applied a non-linear optimization on the vertices to increase the path's length and smoothness.

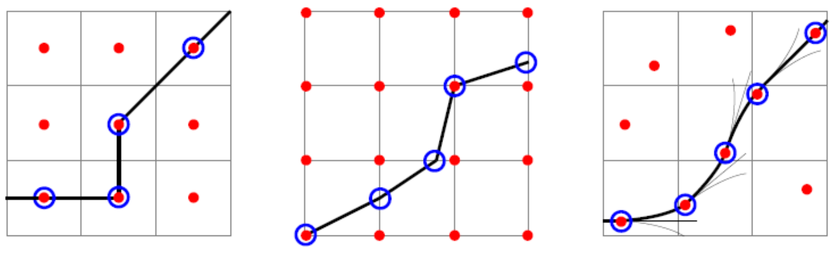

FIGURE 3.1: Graphical comparison between the search methods.

In the last years, autonomous vehicles becomes one of the most interesting areas for researchers. The automated driving tasks include several main tasks such as object detection, road segmentation, path planning, etc. The conventional algorithms are based on cameras that provide the data as images. The efficiency of these algorithms is subject to environmental conditions such as lane marking and light conditions. In this context, the researchers in papers [BH18] discussed the road segmentation task to determine the derivable area of the car by implementing a convolution neural network using LiDAR. The proposed method can be divided into three main steps, prepossessing, neural network processing, and post-processing. In the first step, LiDAR collects the required data from the surrounding environment and provided it to the trained neural network as inputs. In the last step, the results are obtained from the neural network and the required actions are taken. The structure of the suggested neural network is presented in figure 3.2

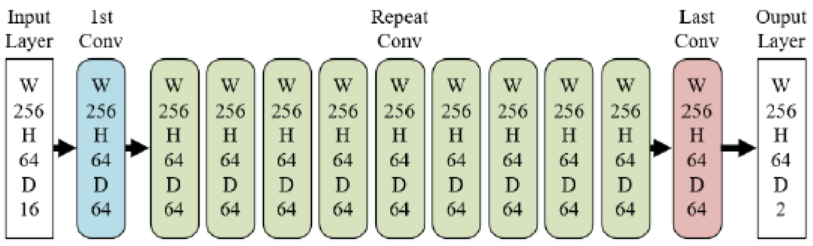

FIGURE 3.2: Architecture of the convolutional neural network for road segmentation [BH18].

Uncertainty and noise have a significant impact on the systems. The uncertainty that varies within a range of data is called heteroscedastic. Providing an accurate estimation is the greatest challenge of the systems that use probabilistic filters to estimate the probability distributions. Kalman filter is an optimal estimator that fuses the information of two sources in the covariance uncertainty matrix. In the paper [RR21], a new method is provided to obtain an accurate uncertainty covariance matrix for the task of tracking the velocity and position of the vehicle. The proposed approach is based on a deep neural network that is trained using two different methods. In the first method, the suggested DNN is trained to directly predict the distribution of the training data using the Gaussian density loss function, while it is trained using the error of the Kalman filter instead of the loss function. The obtained results show that the new approach using DNN provides a great improvement compared to the conventional approach of the Kalman filter. Deep neural networks are efficient solutions across a wide range of high-level applications. Increasing the environment complexity is aligned with increasing the DNN size in order to manage the massive amount of input data, which in turn leads to an increase in computational loads and power consumption. Constraints on size, memory, power, and other resources have an impact on how well-embedded systems that use DNA-based methods perform. In this context, there are several hardware architectures solutions were provided, such as The systolic array architecture which was provided by Google's TPU and colleagues in both academia and industry. It achieved a very high performance compared with conventional architectures. It is a parallel computing architecture that can perform tasks at a high frequency with a small amount of data transferring. The main problem with this architecture is that the systolic array has fixed shapes overall the implementation which means, it is not scalable. Near-data processing is another hardware architecture that allocates the processing unit near to the data and places DRAM memory on the chip which in turn improves the data moving efficiency. Generally speaking, hardware accelerators provide efficient solutions to bridge the gap between artificial intelligence implementations and the hardware platforms in the embedded systems area. An autonomous driving system can be implemented on hybrid computational technologies GPU-FPGA, where the GPU's primary job is self-driving and the FPGA's job is to perform some subtasks such as pedestrian and traffic light detection [HSJ+19]. An end-to-end solution for the design of self-driving cars based on the Xilinx PYNQ-Z2 board is proposed in [WLJ19]]. The most important characteristic of this design is the presence of the Data Process Unit that accelerates the deep learning process. FPGA is faster than the CPU and consumes less power than the GPU when it comes to accelerating the CNN Networks process [AMM+18]. The proposed architecture permits to reduce the execution time and energy consumption compared to the CPU.

A few years ago, FPGA designs were quasi-exclusively performed by electrical engineers since the implementation tools were low levels of abstraction. It was hard to design complex solutions using Hardware Description Language, thus the increase of the design productivity gap. The considerable progress of the implementation tools including reusable IPs increases the popularity of FPGAs allowing a large variety of engineers (from hardware to software engineers) to develop FPGA-based systems. There are many ways to deploy algorithms on FPGAs:

HDL: Using VHDL (Very High-Speed Integrated Circuit Hardware Description Language), an implementation of a vehicle collision avoidance solution has been proposed considering a PID (Proportional–Integral–Derivative) controller based on fuzzy logic [IAZB08]. A HW/SW (Hardware/Software) co-design to implement an Advanced Driver-Assistance Systems was presented in [MCGZCM19]. The hardware accelerator, performing an ANFIS (Adaptive Neuro-Fuzzy Inference System) clustering in the Programmable Logic side of the Zynq SoC, was implemented in VHDL.

HLS (C/C++): Vivado HLS (High Level Synthesis) has been used in [WKA+18] to deploy C/C++ code into a Xilinx FPGA to implement a white line detector where the single line search method gives better results than Hough transform in terms of processing time.

Python:One of the latest trends on FPGA development is the use of Python. The authors in [HWA19] proposed a software/hardware co-design for lane line detection (FPGA/Python implementations). To evaluate the efficiency of the suggested method, the author implemented and executed the application using only software (OpenCV) without hardware deployments. The results show that the execution time of FPGA/Python is 6.34 times faster than software execution, considering the implementation of median blur, adaptive threshold, and accumulation of Hough transform.

MicroBlaze soft processor: A soft processor is an intellectual property core that is constructed utilizing the FPGA's logic primitives. MicroBlaze™ is Xilinx 32/64-bit soft processor core with a rich instruction set optimized for embedded application [XIL]. The classification of traffic signs is an important aspect in autonomous driving. [SGH15] is presented an implementation method for traffic sign detection using HW/SW codesign. The classification is processed on a MicroBlaze soft core and the prepossessing is performed on hardware.

PS+PL vs. MB+IP: Xilinx SoCs are made of two major parts, PS (Processing System) and PL, which can be exploited to have HW/SW codesign based systems. The authors in [HVPO15] show that implementing a TSR (Traffic Sign Recognition) system on a Xilinx SoC gives better results than its implementation in an MicroBlaze + IP based system.

XSG (Xilinx System Generator): XSG or Model Composer is a powerful development tool to implement FPGA designs as it is suitable for Digital Signal Processing applications. The authors in [HAM18] used XSG to model, simulate and implement a traffic signs detection system which presents promising results.

The accuracy and delay of autonomous driving systems have a major impact on how the vehicle handles the surrounding environment. To make the delay, resulting from data inputs, deterministic a solution is proposed in [AW18] considering bypassing the CPU from the input data path figure 3.3

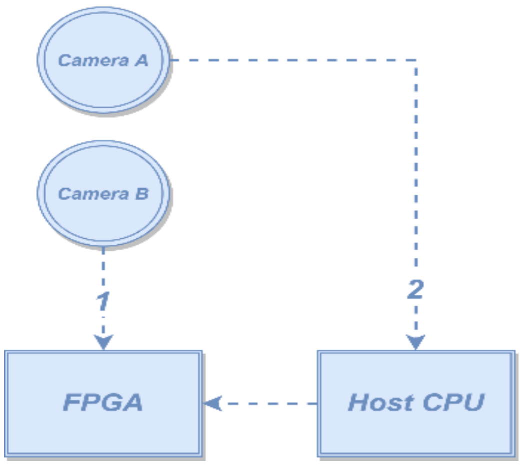

FIGURE 3.3: Bypassing the CPU from the input data path [AW18].

The biggest challenge when implementing AI algorithms in FPGAs is the difficulty of the hardware design. H. Bingo [Bin18] proposes a solution to this problem applying PYNQ board which allows the use of Python for its wide range libraries, especially in AI and Image processing. O. Chang-song and Y. Jong-min [sOmY19] discuss the most popular technologies to accelerate deep learning processes for autonomous cars industry. Some steps are introduced to avoid collisions in the air for UAV (Unmanned Aerial Vehicle) using four cameras based on FPGA SoC [KZN19]. The collision avoidance algorithm has already been implemented in GPU-based system, but the solution is not practical as it requires high energy. Road segmentation is a very important process in self-driving cars which identifies and describes the drivable parts of the roads. The problem with this process is the need of high computational resources. A model based on FPGA is suggested in [LBH18] to perform a real-time and low power road segmentation. The authors suggest the use of LiDAR than the use of traditional cameras since they suffer from image unclarity in poor lighting conditions. FPGA is used for scene perception based on CNN. It took about 16.9 ms to accomplish a CNN operation using Xilinx UltraScale XCKU115 FPGA. In self-driving cars environment, if there is more than one car starting from the same place and going to the same destination, the decision-making system will choose the same optimal path for all cars. There will be one busy road/track and other uncrowded. To solve this issue, paper [DHa+19] proposes a solution implemented on FPGA applying game theory. This solution relies on direct communication between vehicles to apply game theory instead of using the cloud. The author of [WHA18] relies on the use of two FPGA board. PYNQ-Z1 to deal with detecting obstacles and pedestrians by implementing light weight BNN (Binarized Neural Networks) and Zynq-Xc7Z010 to recognize traffic lights and detect road lines. The two FPGA boards were connected to each other using Ethernet to be able to communicate. The obtained results show an improvement in terms of execution time and the accuracy of identifying the obstacles. Generally, designing machine learning-based models goes through two main steps, training and inference phases. In [LOG+18] a hybrid methods is suggested, and a machine learning platform was built so that the training phase takes place in the GPU and inference phase in FPGA as figure 3.4 shows.

In [ZWZ+18] a tool is provided to accelerate and improve the performance of applying Deep Neural Network (DNN) on FPGA. The proposed design used two different methods to manage the on-chip FPGA memory. The first one is to store the complete feature maps on the chip in order reduce the complexity of the data movements. The second technique is to automatically generate buffers to overcome the data shortage when the processes are fetching data from the external memory. The proposed design provides real-time performance for processing HD videos and deliver a higher efficiency (up to 4.35x) than the GPU-based solutions. The article [Ahm20] presented multi-core processing units capable of increasing the level of automation in Urban Air Mobility (UAM) aircraft by using a parallel sensor fusion structure to increase accuracy and safety during

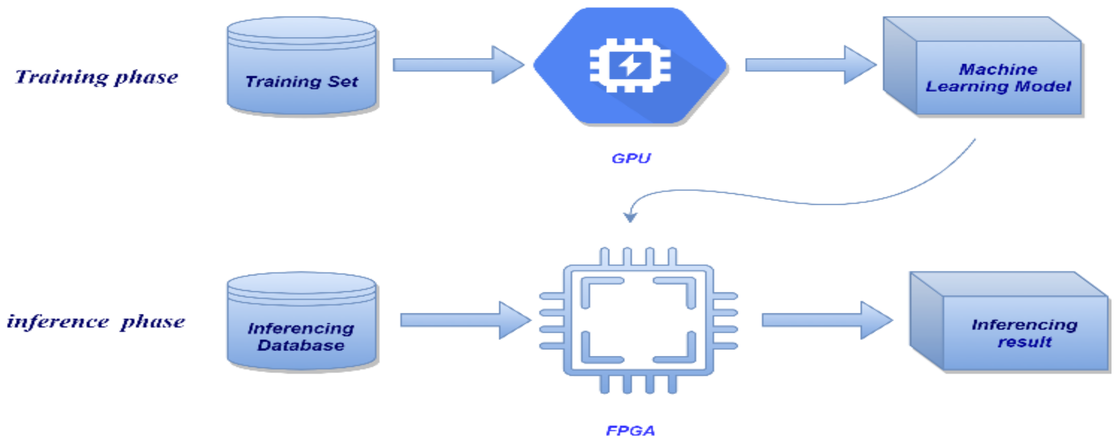

FIGURE 3.4: The architecture of the hybrid machine learning platform [LOG+18].

landing. These processing units are based on FLS (Fuzzy Logic System), Which interacts with several sensors that have different in frequency ranges and spatially separated. The proposed method was designed and tested using MATLAB and VHDL on Intel Altera OpenVINO FPGA board. Paper [KVRB20] presents an FPGA-based method for optimizing LSTM functionality in a critical-time environment. The proposed architecture gives a similar output in quality compared to a traditional LSTM, but in terms of speed it is 415 times faster. This method proposes an approximate computing scheme that enables the calibration of the QoR (Quality of Result). This model approximates the weight arrays in a neural network using low-rank SVD (Singular-Value Decomposition) in addition to the distribution of weights arrays based on the importance of their components. These techniques allow to restructure the calculations for LSTM and design computing system that carries out the most important operations first, to get peak QoR within time constraints.

3.3 Chapter Summary

In this chapter, I introduced the latest proposed FPGA-based solutions to accelerate AI processes dedicated to autonomous vehicles. It is doubtless that reaching fully autonomous and safe cars requires overcoming many challenges. For instance, real-time processing of the huge data coming from multiple sensors considering the energy consumption in addition to the safety issues. Exploiting reconfigurable computational technologies significantly enhance the performance of the solutions that especially involve artificial intelligence and autonomous vehicles since they require parallel computing. Depending on the solutions discussed in this work, we are about to witness fully autonomous and safe vehicles in the next few years considering embeddability requirements.

Chapter 4

Model Predictive Control for Autonomous Vehicle Steering System and FPGA Implementations

The autonomous vehicle steering system, a multi-input multi-output (MIMO) system, is challenging to design using traditional controllers due to the interaction between inputs and outputs. If PID controllers are used the control loops are executed independently of each other as there is no interaction between the loops. Designing a larger system increases the controller parameters requiring tuning. Model Predictive Control (MPC) overcomes this problem, as it is a multi-variable control method taking into account the interactions of the variables in the target system. Achieving a high safety level is also critical for autonomous vehicle systems. This can be provided by an MPC controller, which can handle constraints such as maintaining a safe distance from other cars. The wider applicability of the Model Predictive Controller calls for more efficient hardware architectures for implementation. The aim of this work is to achieve optimal implementation of the MPC controller by increasing the computational speed in order to reduce execution time for optimization. An MPC controller is used to control the steering system of an autonomous vehicle to keep it on the desired path. A traditional MPC controller is used to control the system where the plant dynamics do not change, whereas an Adaptive MPC controller is used when the system is nonlinear or its characteristics vary with time (the longitudinal velocity changes as the vehicle moves). Results are discussed in terms of performance, resource utilization, cost, and energy-effective implementations taking into consideration a reasonable size number of constraints handled by the controller.

4.1 Introduction

In recent years, research in the automotive industry has been growing in order to address the challenges of this application domain. Automotive control applications require high performance and cost reduction at the same time [UDoT18]. The control system requirements are becoming higher, and to improve control performance, the optimization process is incorporated into the control system design. The optimization process is subject to an increased number of factors, such as physical, safety, and economic constraints (power consumption, actuator saturation, etc.). In this context, Model Predictive Control is a powerful optimization strategy for feedback control based on the model of the system. Basically, an MPC controller runs a set of forecasts forward in time on the system model for different actuation strategies. MPC determines the immediate next control action based on the optimization. Next, it reinitializes the optimization in order to define the next control input [PBT$^+$08]. The current and future control inputs are determined based on minimizing the difference between the target set-point and the predicted output [LYM06]. MPC features and capabilities are very effective in terms of meeting the requirements and achieving the optimization tasks. A basic MPC controller solves Linear Programming (LP) problems, which can be formulated as a quadratic programming (QP) problem [LWM08]. Also, the MPC controller has a natural capability to handle soft and hard constraints. That means the requirements that are imposed by the operating conditions can be managed and formulated using the constraints. However, MPC controller implementation has several challenges such as high computational load and high power consumption, whereas the embedded system applications have limitations in their hardware resources.

One of the most effective solutions, in order to achieve MPC implementations for embedded system applications that have constraints related to the computational time, is the use of hardware acceleration. In this context, the deployments of an embedded MPC controller can be achieved using reconfigurable hardware such as Field Programmable Gate Array or System on Chip, which is popular due to its high computational capabilities, parallel processing and development framework [LYLM09]. In this context, the main contributions of this paper can be summarised in two main points. First, studying, analysing and improving the implementations of the MPC controller for the task of automated driving especially with changing dynamic systems. Second, the use of rapid prototyping method (hardware/software co-design) to deploy the design on FPGA (hardware-embedded system). I used three different optimization strategies to optimize the deployment considering performance, execution time and resources consumption. These strategies involves Logical optimization, placement of logic cells, and routing the connections between cells. The research applied functional on-target rapid prototyping using Embedded Coder and HDL coder. The suggested implementation method is based on taking the optimization problem of the control method through MATLAB Simulink, Fixed-Point Designer, Embedded Coder, and HDL Coder. The suggested method allows the authors to focus on the verification, validation, and test of the embedded system rather than programming, which in turn gives the ability to refine the design, tune the MPC controller parameters and see the results in the real time. Finally, different optimization strategies were implemented and the obtained results were compared in terms of reducing the execution time and hardware resources consumption.

FPGA-based systems have been applied for a variety of applications, such as image and signal processing, aerospace, energy, autonomous vehicles, telecommunications (5G), and the medical field. In paper [AG11] an analytical study for an Adaptive MPC controller under external disturbances signals was provided, the Lipschitz-based approach was used and provides satisfactory stability and robustness. Saragih et al. used the MPC controller for a visual-based control system application (face tracking system) to control the motion of a robot, where the MPC controller was implemented to control the camera movements in order to keep the tracked face at the center of the camera – see [SKM+19]. Paper [CK18] provides an overview of a real-time optimization problem for automotive and aerospace applications with a focus on MPC controllers. The optimal control problem was formulated based on the cost function (J) and the system constraints, in addition, numerical algorithms and their implementations on an embedded computing platform were discussed. The improvement of fuel economy for power-split hybrid electric vehicles (HEV) was discussed in [BVP+11]. The energy management system was formulated as a nonlinear and constrained system. The MPC controller was used to split the power between the combustion engine and electrical machines at the different system operating conditions. The proposed approach provided an improvement compared to the controllers in commercial Powertrain System Toolkit (PSAT) software. The research reported in [FTAH08], proposed a control approach based on combining steering and braking MPC controllers. The authors in the paper introduced two model predictive controllers. The first one was implemented on a four-wheel vehicle model which determines the steering angle and braking torques to track the desired trajectory. The second MPC controller was implemented on a simplified bicycle model with a smaller number of inputs. The obtained results showed that the first controller provides good performance in terms of tracking the reference trajectory at low and high speed, but the computation was time-consuming. On the other hand, the second controller showed unsatisfactory performance at high speeds due to the simplicity of the vehicle model [FTAH08].

Paper [STAK18], presented research of edge cloud on the Internet of Things (IoT) where the Model Predictive Controller evaluates the system properties. The paper presented the potential of merging the IoT, 5G, and cloud computing with the efficiency of deploying the automatic control system for time-sensitive and mission-critical processes. Haidegger et al. in [HKP+11] stated that the predictive and model-based control gives satisfactory performances only in the case of providing the accurate system's behavior and cascaded control approach. An empirical design with the use of the Smith predictor for a telesurgical robot system was suggested in order to deal with the large latencies. In the same context as the paper [HKP+11], the article [TKP+11] suggested a cascaded control structure to deal with the time delay in a teleoperation robot system. The suggested method used the extended Kessler's method sported by a predictive control method. A fuzzy–PID controller was also suggested to improve the performance. Using the extended Kessler's method with the Smith predictor provides good control. MPC controller deals with the linear-time-invariant (LTI) plant model, which allows for predicting the future behavior of the system [WIP02]. Nevertheless, the paper [NG19] suggested a strategy to control heterogeneous traffic flow. Linear Parameter-Varying (LPV) model was suggested where the model deals with a non-linear traffic flow system that contains autonomous and human-driven vehicles with different operating conditions. LPV provides the ability to control the nonlinear system which uses different linear controllers for different operating points. LPV model uses a scheduling variable to enable the controller based on the current operating point of the system [Eig17]. Our work discusses the use of an MPC controller for an autonomous vehicle steering system and its implementation using MATLAB Simulink and an FPGA board. The implementation of FPGA is conducted using an HDL coder. In our work, the contributions are presented in two main points, providing a methodology for adaptive MPC development for the changing dynamics system and optimizing the hardware deployment (FPGA) using different optimization methods, including logical optimization, placement of logic cells, and routing the connections between cells.

4.2 MPC and Adaptive MPC Working Principles

In a control problem, basically, the goal of the controller is to calculate the input variables to the plant so the plant responds in a way that makes its output track the reference output. Figure 4.1 shows the standard control loop diagram.

4.2.1 Model Predictive Controller

Model Predictive Control uses a future prediction strategy in order to calculate the input. To ensure that the output of the plant follows the target reference output, the MPC controller uses what is called an optimizer. The prediction strategy is based on

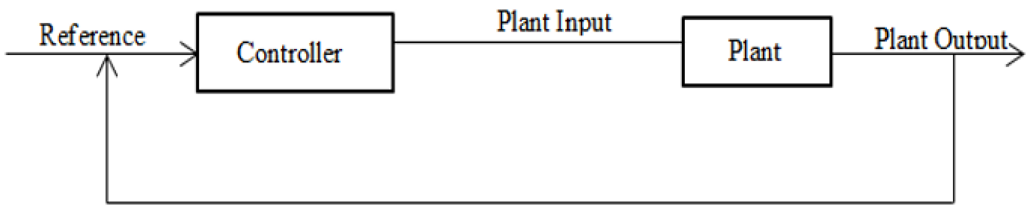

FIGURE 4.1: Standard Control Loop

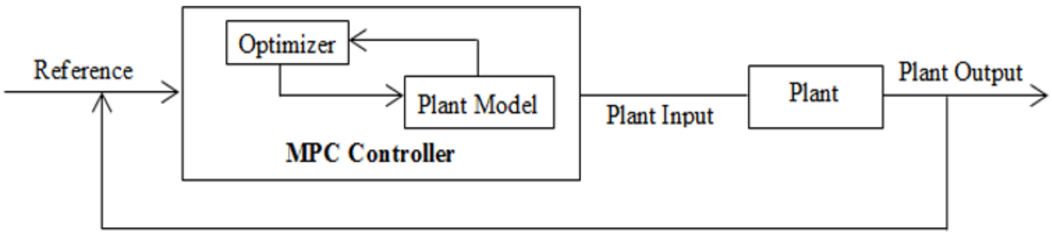

FIGURE 4.2: Traditional MPC control diagram

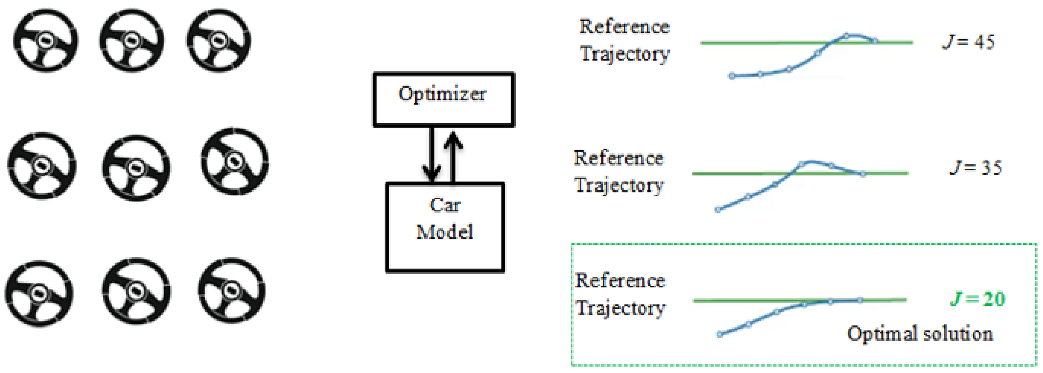

FIGURE 4.3: Future Prediction Strategy for Optimization Problem

the use of a plant model (car model) by the MPC controller to simulate the car's path in the next P time steps, where P is the prediction horizon which represents the time, the MPC controller looks forward in the future to make the prediction. The Model Predictive Controller simulated different future scenarios in a systematic way, and here the optimizer comes to the picture by determining the best scenario which achieves the minimum error between the reference and the predicted trajectory. The minimum error corresponds to the minimum cost function, which means the scenario of the predicted trajectory with the minimum cost function provides the optimal solution. Figure 4.2 shows the traditional MPC controller, and figure 4.3 shows a future prediction strategy, where each scenario represents a series of steering wheel movements in order to follow the reference trajectory, and as mentioned above the optimal scenario is the one that achieves the minimum cost function [Mat18a], [NVR13].

The scenario with the minimum cost function J = 20 is the optimal solution, which achieves the optimal reference trajectory tracking. The design presented in this article proposes that the new state of the car model can be measured, while in the case of the state model cannot be measured. The MPC controller uses the so-called "state estimator" to estimate the state of the system and feed it back to the controller. The MPC controller uses static Kalman Filter (KF) in order to update the controller states (plant model states, measurements noise model state and disturbance model state).

4.2.2 Adaptive Model Predictive Controller

The traditional MPC controller is unable to deal with the changing dynamics systems effectively since it uses a constant internal plant. When the system is nonlinear or its conditions vary with time, the accuracy will be negatively affected and the performance becomes unacceptable. To deal with these systems, an Adaptive MPC (AMPC) controller is used. AMPC controller handles the changes in operating conditions by providing a new linear model at each time step to achieve accurate prediction for the new conditions, as shown in figure 4.4 [Mat].

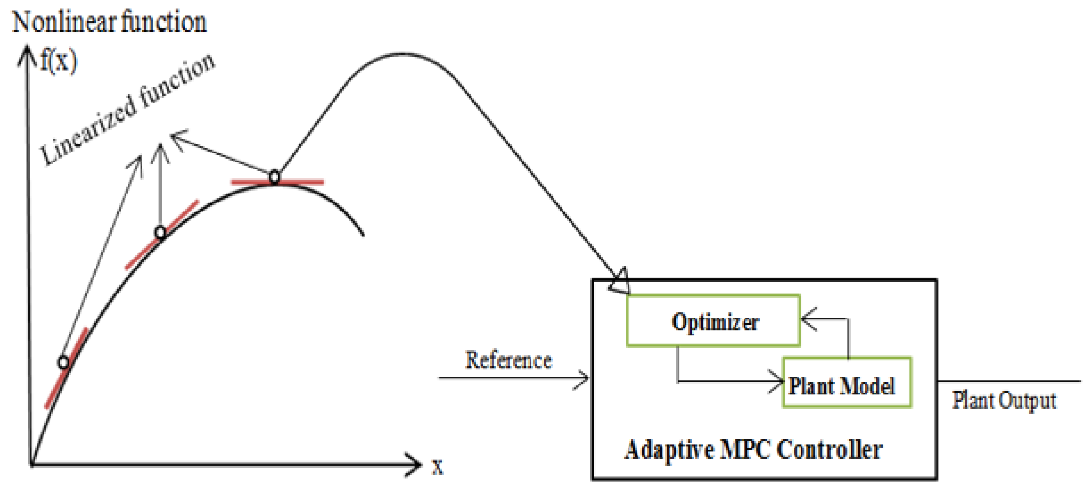

FIGURE 4.4: Adaptive MPC Controller

The optimization problem in the Adaptive MPC controller remains the same, which means the same number of states and constraints for the varied operating conditions. The Adaptive MPC controller requires a discrete plant model, which means, the continuous-time state space needs to be converted to discrete-time (zero-order hold method). The Adaptive MPC Controller receives the updated discrete-time state space containing the following [AG11]:

A: n_x by n_x matrix signal, where n_x the number of plant model states.
B: n_x by n_u matrix signal, where n_u the total number of plant inputs.
DX: Vector signal of length n_x.

where DX is computed by equation 4.1, which provides the updated discrete-time state, where u_k and x_k are respectively the inputs and the state values for the current time step k

$$DX = A(u_k) + B(x_k) - x_k \tag{4.1}$$

4.3 The optimization Problem

The MPC controller solves an online optimization problem, which is a Quadratic Problem (QP) for specific at each control interval. Figure 4.5 shows the control algorithm of the Model Predictive Controller.

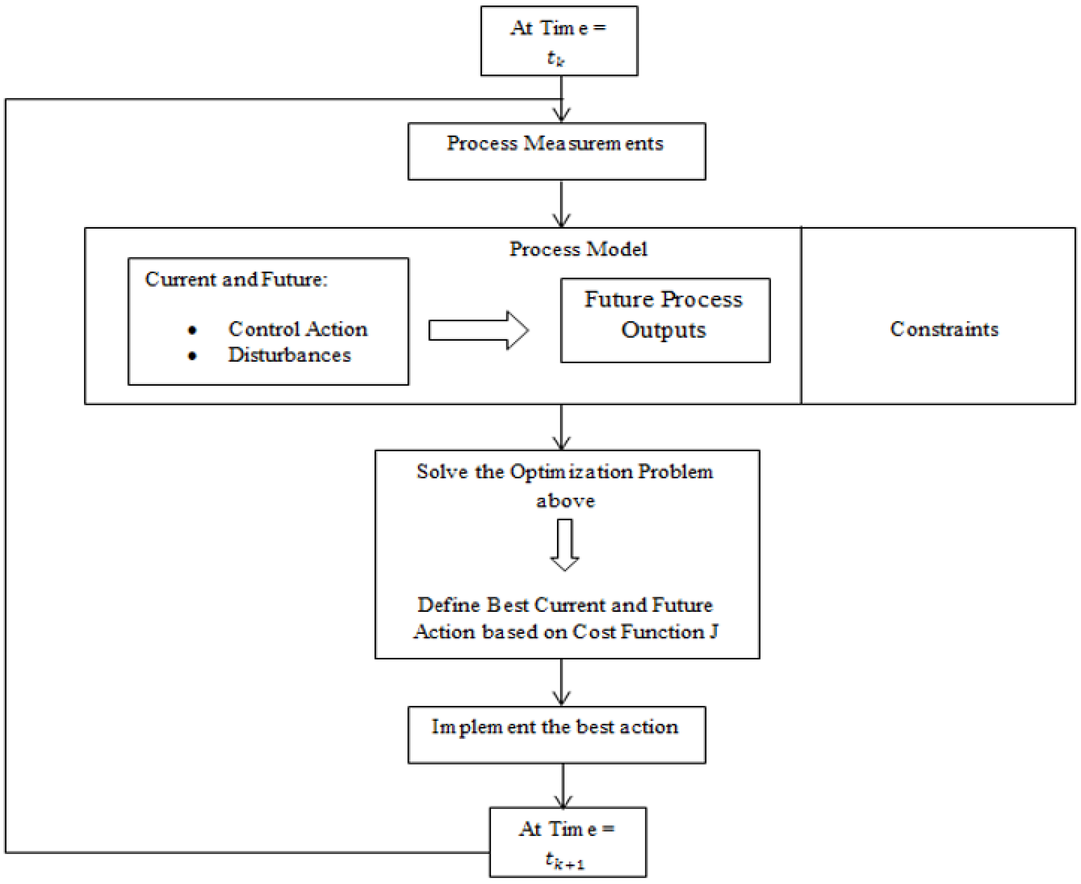

FIGURE 4.5: MPC Control Algorithm

The optimization problem includes the followings:
Cost Function: also called objective function, it measures the controller performance, and the goal is to be minimized.
Constraints: It represents the soft and hard constraints which must satisfy the system conditions such as the physical bound. To achieve the optimization, the MPC controller needs to calculate the control inputs driving the output of the plant that are very close

to the desired reference. This process is performed in a systematic way by applying different scenarios and minimizing the cost function of the optimization problem. The cost function J of the autonomous vehicle's steering system can be formulated as [Mat]:

$$J = \Sigma_{i=1}^{p} w_e e_{k+i}^2 + \Sigma_{k=0}^{p-1} w_{\Delta u} \Delta u_{k+1}^2, \tag{4.2}$$

where w_e is the weight of the predicted error e_{k+i} and $w_{\Delta u}$ is the weight of the steering angle increments Δu_{k+1}. Cost function goals are to minimize both, the error between the predicted trajectory and the reference and the change in the steering angle between the consecutive time steps. The optimal solution corresponds to the smallest value of the cost function.

Decision: Modify the manipulated variables in order to achieve the minimization of the cost function and to satisfy the constraints. The MPC controller computes the manipulated variable by solving the quadratic problem using a custom QP solver which in turn converts the linear optimization problem to the general form of the QP problem.

4.4 Model Predictive Controller Design Parameters

Designing the MPC controller takes into consideration the required constraints such as the steering angle limits. Figure 4.6 presents the main parameters and terms of the MPC controller, where the following nomenclature applies: k is the current sampling step and T_s the Control Time Step. Prediction horizon (P): number of time steps (the time on which the MPC controller looks forward to the future to make the prediction). Control Horizon (M): number of the possible control moves to time step k+P. The design parameters of the MPC controller are very important as this affects the performance and the computational complexity of solving the optimization problem. The choice of the design parameters should achieve the balance between the computational load and the performance. There are general recommendations, which can be taken into consideration for the parameters.

Sample time (T_s): determines the rate that the controller executes the control algorithm. In the case of Control Time Step T_s interval is too long, the controller will not be able to respond in time to the disturbance, which means that the performance will be negatively affected. On the other hand, if Ts is too short, the controller's response will be faster, but this causes a significant increase in computational load. The recommendation, in this case, is to choose T_s between 10 to 20 samples of the Rise Time T_r in an open-loop system, where T_r is the required time that the response takes to rise from 10% to 90% of the steady-state as figure 4.7 shows [Mat18a].

Prediction horizon : should be chosen in a way that covers the dynamic changes of the system and the recommendation are to choose P to have 20 to 30 of samples covering the open-loop transit system response [Mat18a], [NVR13], [Wor12] and [YZO16].

Control Horizon: Only the two control moves have a significant impact on the response behavior, choosing a large control horizon will only increase the computation complexity, based on that, the recommendation is to choose M to be 10 to 20 of the prediction horizon. A small value of M provides stability while in contrast, large values reduce the robustness. It is recommended to choose M to be between 3-5 – as presented in [GS10], [Mat18a], [NVR13], and [TMT06].

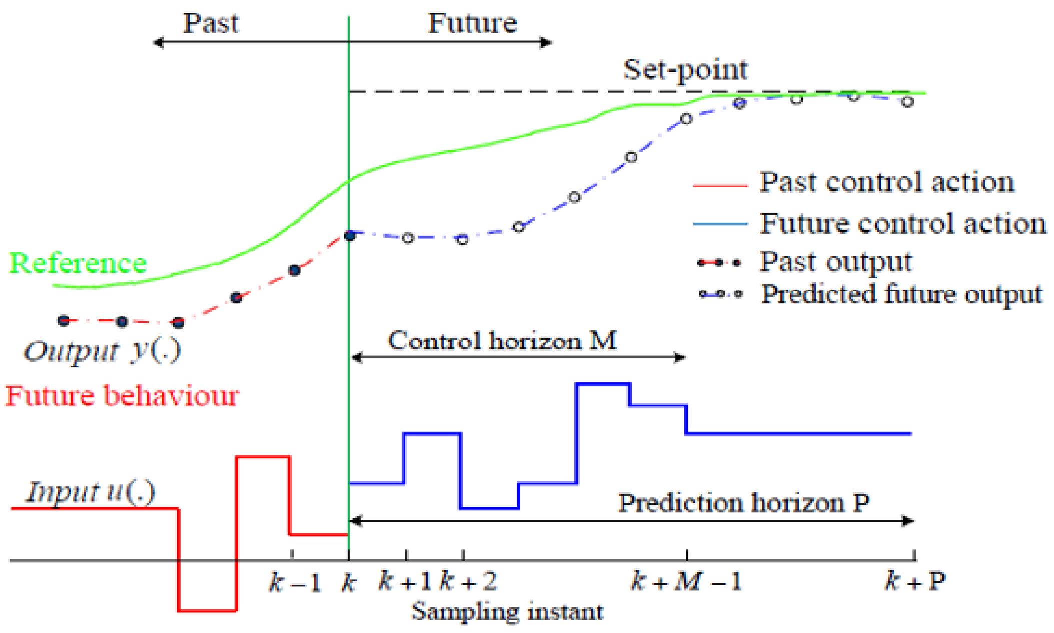

FIGURE 4.6: MPC schema for the main terms [YLLL17]

For the model I suggested in the work, the following strategy was used in order to choose the parameters that achieve satisfactory control performance: First, I initialized the parameters based on the recommendations above, regarding the Sample Time, Prediction Horizon, and Control Horizon. Next step, is about tuning the parameters and then evaluating the MPC controller performance using the MPC Designer MATLAB toolbox until the optimal values provided the best control performance were determined. The weights of the inputs and outputs were determined using the MPC Designer by setting nonzero values to the inputs and outputs which need to track a reference value. Based on that, the weight equal is set to zero for the steering angle as it does not track a target. The weight of the lateral position and yaw angle were determined with nonzero values as the main objective is position tracing.

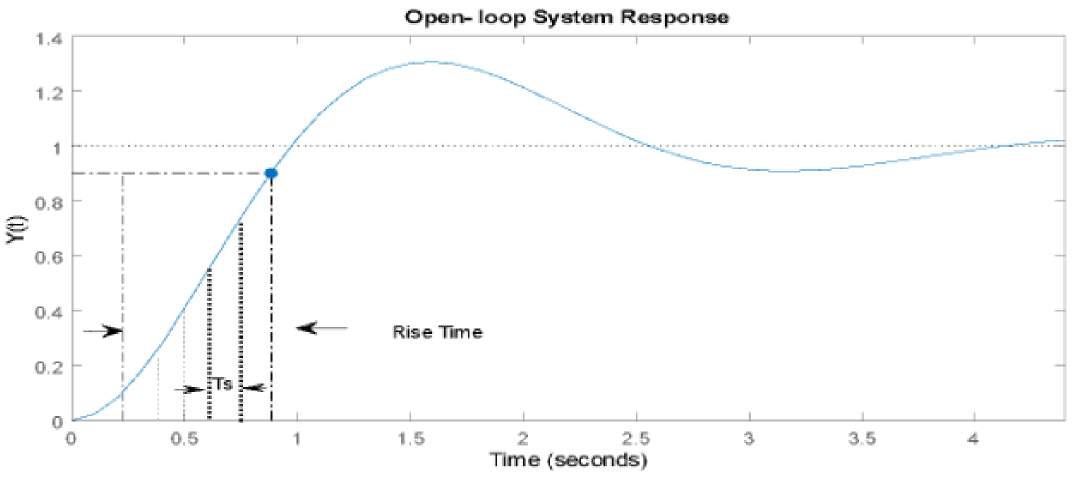

FIGURE 4.7: Control Time Step $T_s$and Rise Time T_r

4.5 The Vehicle Model

MATLAB MPC designer application was used to design the controller that steers the vehicle autonomously. Figure 4.8 shows the global position of the vehicle in X and Y axes where (X, Y) are the vehicle's global position, v_y is the lateral velocity and v_x is the longitudinal velocity. The parameters that need to be controlled are: Yaw angle θ and the front steering angle δ. The state-space of the model is given by the following equations:

$$\frac{d}{dt}\begin{bmatrix} v_y \\ \theta \\ \omega \end{bmatrix} = \begin{bmatrix} -\frac{2c_{af}+2C_{ar}}{mv_x} & 0 & -v_x - \frac{2c_{af}l_f-2c_{ar}l_r}{mv_x} \\ 0 & 0 & 1 \\ \frac{2l_fc_{af}-2l_rc_{ar}}{I_zv_x} & 0 & \frac{-2l_f{}^2C_{af}+l_r{}^2 2C_{ar}}{l_zv_x} \end{bmatrix} \begin{bmatrix} v_y \\ \theta \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{2c_{af}}{m} \\ 0 \\ \frac{2l_fc_{af}}{I_z} \end{bmatrix} \delta \tag{4.3}$$

$$\dot{y} = v_x\theta + v_y \tag{4.4}$$

where ω is yaw rate, m is the total mass of the vehicle, l_z is yaw moment of inertia of the vehicle, l_f and l_r are the longitudinal distance from the center of gravity to the front tires, c_af is cornering stiffness of tires and y is the lateral position.

MPC controller performs all the calculations using discrete-time state space. When a plant model is specified for the MPC controller, the following needs to be performed [Mat]:
Conversion to state space: the model is converted to linear time-invariant state space model.

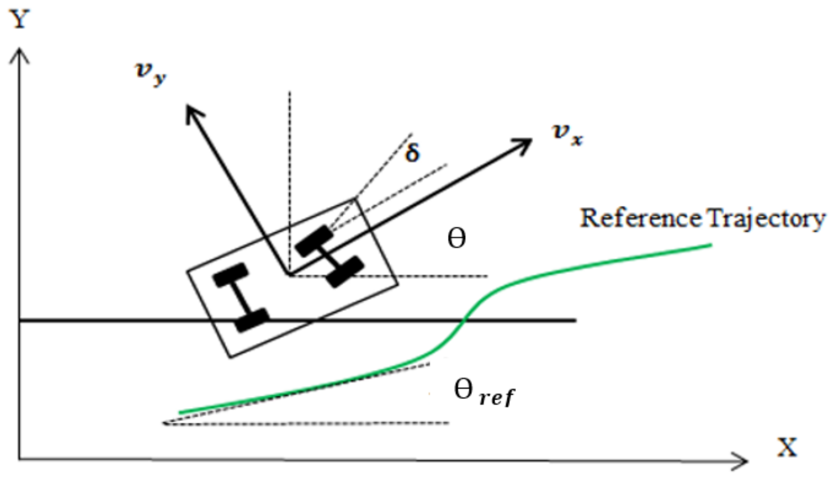

Figure 4.8: The global position of the vehicle

Discretization or resampling: in the case of different sample times between the model and the MPC controller the following occurs:

- In the case of a continuous model, it must be converted to a discrete-time dynamic system model.
- In the case of the discrete model, the discrete-time dynamic system model is re-sampled in order to generate an equivalent discrete-time model with a new sample Time T_S

There are different ways to discretize a continuous model, in the proposed one, the continuous-time dynamic system model was discretized using zero–order hold on the inputs and sample time of T_S. This can be used also for resampling the discrete-time dynamic system model with new sample time T_S.

4.6 Design of the MPC and HDL Code Generation

Based on the MPC control diagram the Simulink model was built. First, the required blocks (Plant model and Reference) were added to the workspace and linked to the MPC controller. The first input of the controller is the measured output and the second one is the reference trajectory, which was created using the Driving Scenario Designer Toolbox in MATLAB. As mentioned before, the MPC controller was designed using MPC Designer, where the internal plant model and the scenario are defined and the designing parameters such as sample time and control horizon were set using the strategy defined in section (3.4). In addition, the hard and soft constraints and their weights for the inputs and outputs such as the steering angle and the rate of change were set. In the case of an unchanging dynamics system, the input of the vehicle model is the output of the model predictive controller (the steering angle) and the outputs are the lateral position and Yaw angle. Figure 4.9 presents the MPC controller model for linear systems (unchanging dynamics system). On the other hand, in the case of changing dynamics system, the longitudinal velocity is a second input for the vehicle model and the adaptive MPC controller will use the plant mode output (State) to perform the new prediction for the updated model state. Figure 4.10 presents the Adaptive MPC controller model for nonlinear systems (changing dynamics system) with the Update Plant Model block.

Manual coding is time-consuming compared to automatic code generation, which in turn lets the designers focus on verification, validation and testing rather than programming. The model-based design generally provides an effective improvement in terms of system reliability and reduces the total project time by up to 33% and the cost by 20% compared to the traditional methods (handwritten code) [ST13].

The floating-point model needs to be converted to fixed point in order to reduce the

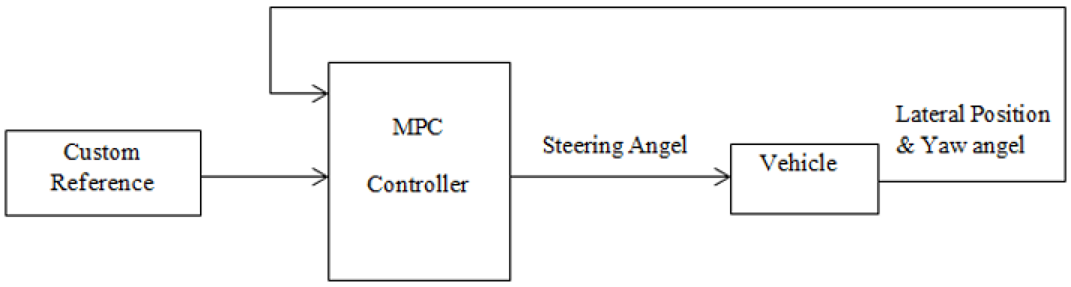

FIGURE 4.9: MPC controller model for linear system (Constant longitudinal velocity)

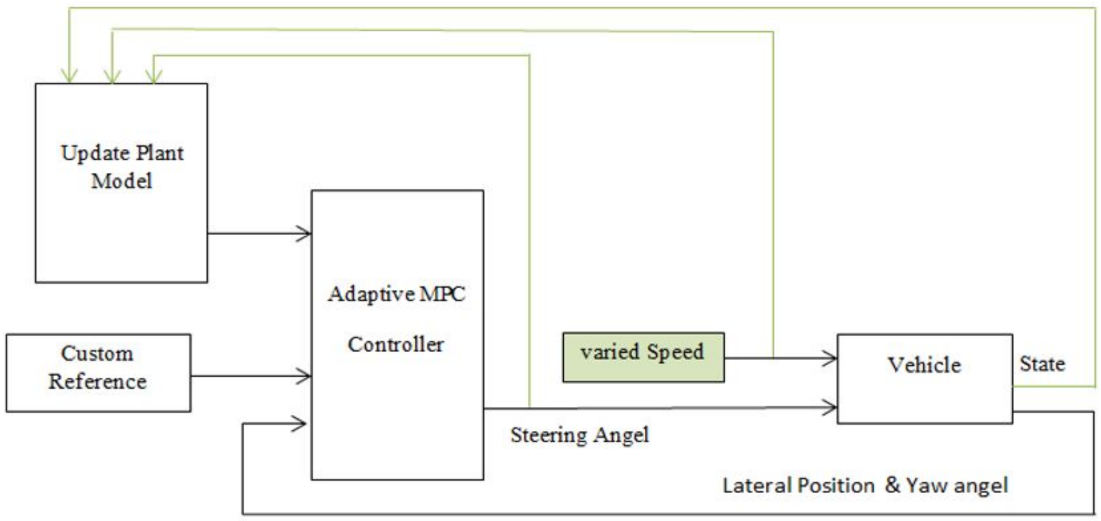

FIGURE 4.10: Adaptive MPC controller model for nonlinear system (varied longitudinal velocity)

hardware resources [OMM+14]. The steering system was designed and simulated using MATLAB Simulink and implemented on System on Chip target using an embedded coder and HDL coder. The working methodology is presented in figure 4.11. First, the MPC controller model was created and the parameters were determined in MATLAB (see table 4.1), followed by the HDL coder model and functional verification. Intellectual Property (IP) was created by Vivado. The MPC controller project was created and the MPC IP was connected to the Processing System (PS) through the AXI interface. Figure 4.12 shows the block design of the MPC system. The next step of the development (see figure 4.11) was the bit-stream generation and export to the software development system (Xilinx SDK). The last step of the development was the software design and test. The generated project in Xilinx SDK together with the bit-stream downloaded and the target FPGA was programmed. In MATLAB Simulink the MPC model and MPC hardware system were tested and checked with Hardware In the Loop (HIL) simulation. The results are presented in the next section

TABLE 4.1: MPC design parameters

MPC controller parameters	
Parameter	Value
Sample time Ts	0.1 s
Prediction horizon	10 s
Control horizon	3 s
Constraints	
Steering angle	[-0.5 , 0.5] rad
Changing rate	[-0.26 , 0.26] rad

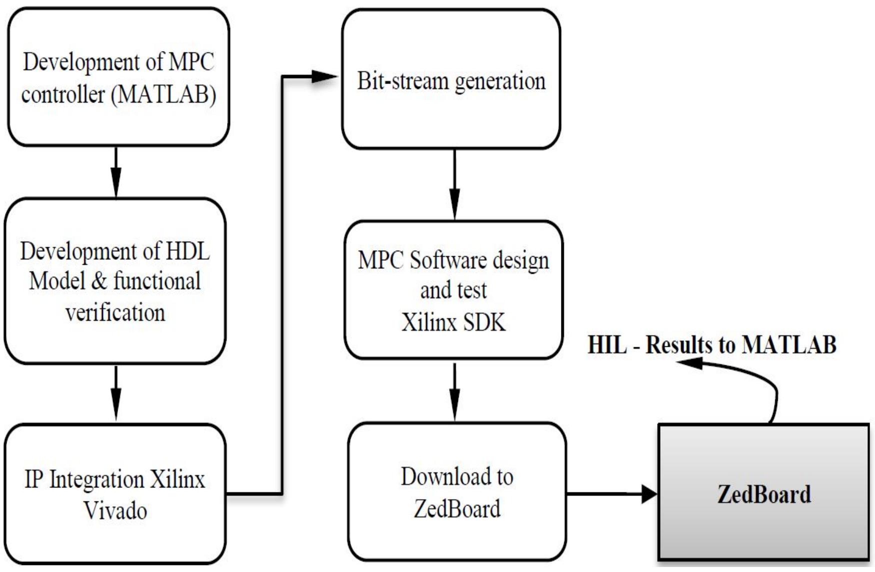
Development of MPC controller (MATLAB)
Development of HDL Model & functional verification
IP Integration Xilinx Vivado
Bit-stream generation
MPC Software design and test Xilinx SDK
Download to ZedBoard
ZedBoard
HIL - Results to MATLAB

FIGURE 4.11: The design workflow of the proposed solution

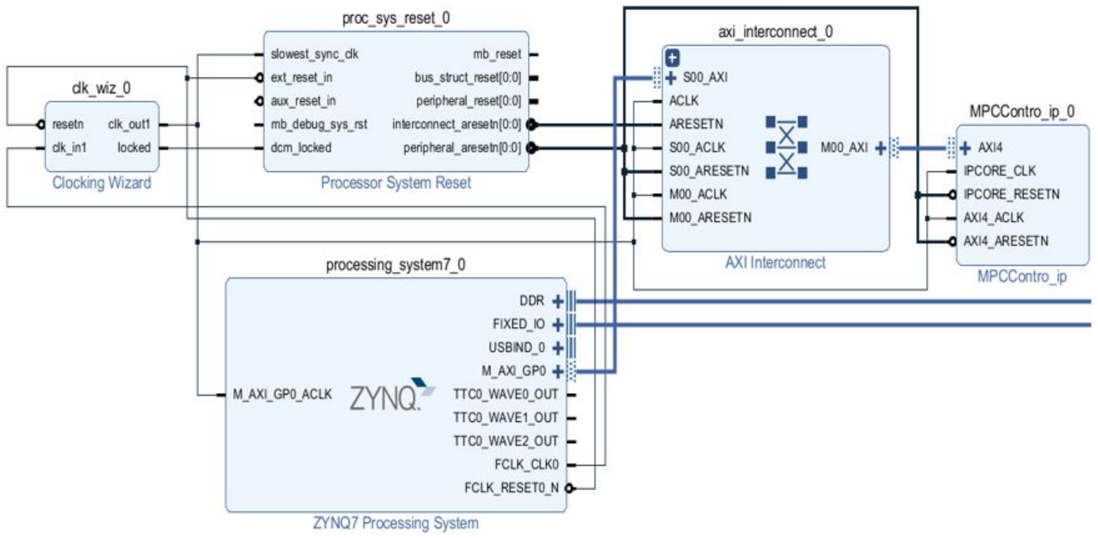

FIGURE 4.12: Vivado Block Design

4.7 Simulation

4.7.1 Matlab Simulink Implementations

The steering system model was tested using MATLAB Simulink for both MPC and Adaptive MPC. figure 4.13 shows the performance of the MPC controller at a constant longitudinal velocity, and figure 4.14 shows its performance at varied longitudinal velocity. The obtained results in figure 4.13 and figure 4.14 show that the MPC controller achieved satisfactory performance for the constant operating conditions, while it failed to handle the system with changing longitudinal velocity. figure 4.15 shows the performance of the Adaptive MPC controller for the changing dynamic system (varied longitudinal velocity). Results demonstrate that using the Adaptive MPC controller for the changing dynamics system yields good performance in terms of tracking the reference (lateral position and yaw angle).

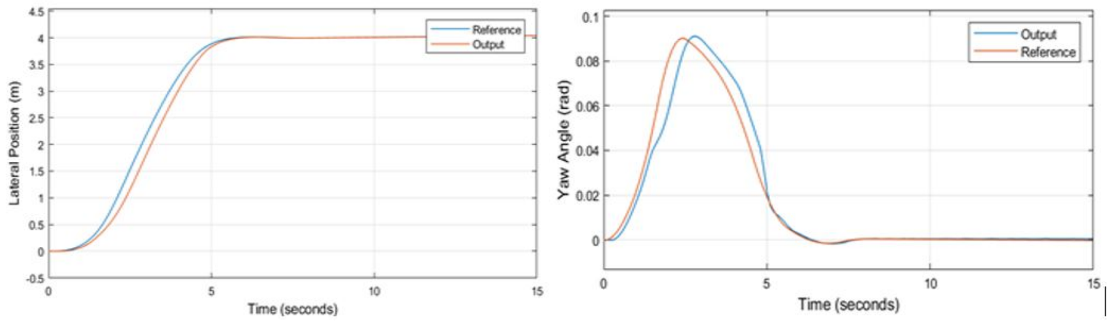

FIGURE 4.13: MPC controller performance at constant velocity

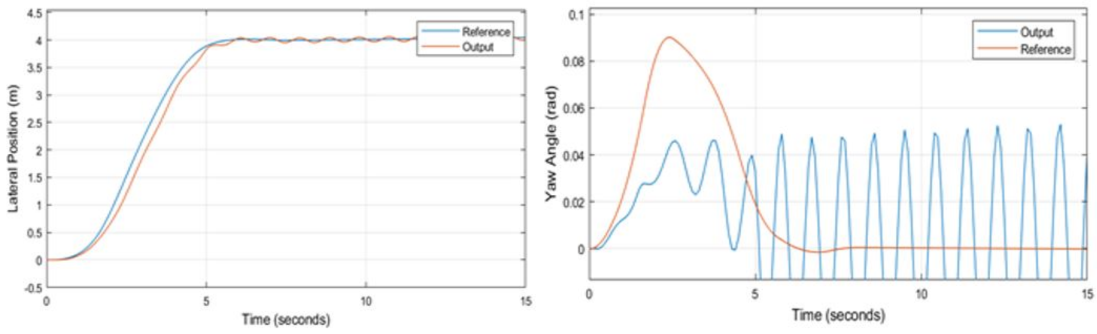

FIGURE 4.14: MPC controller performance at varied velocity

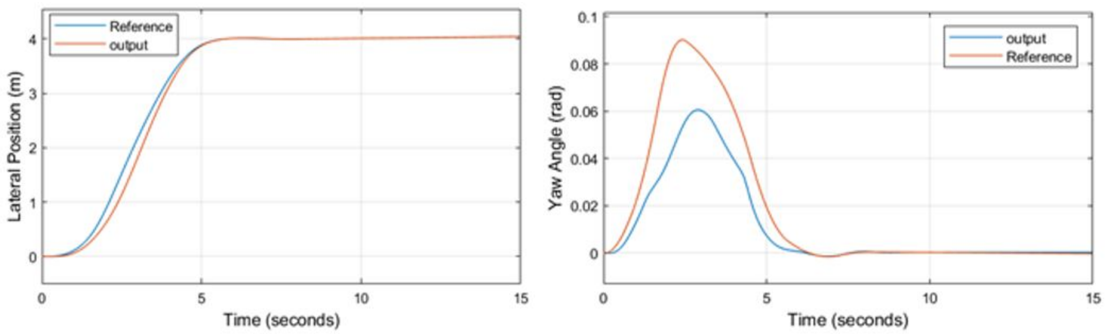

FIGURE 4.15: Adaptive MPC performance at varied velocity

4.7.2 FPGA Implementations

Both models (MPC and Adaptive MPC controller) were implemented on FPGA and the results were compared with the results obtained using MATLAB Simulink. The experiments showed slight differences in terms of performance between the implementations (Simulink and FPGA). figure 4.16 and figure 4.17 show the performance of the MPC controller at constant longitudinal velocity, and the performance of the Adaptive MPC controller at varied longitudinal velocity, respectively. Figure 4.18 and figure 4.19 clearly show the difference in performance between the two controllers' implementation.

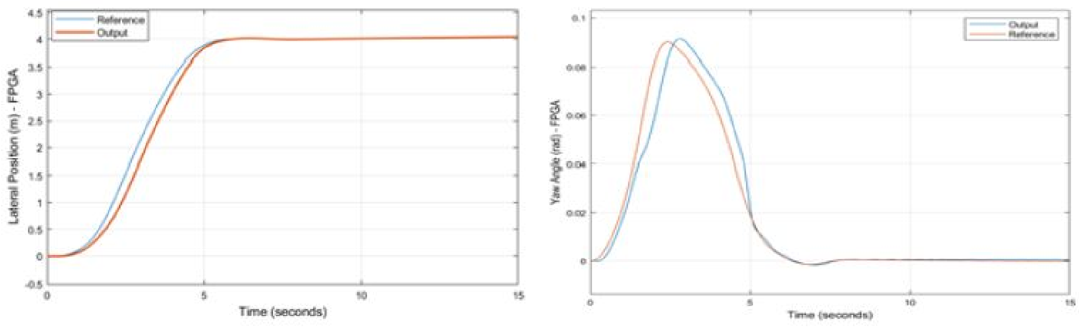

FIGURE 4.16: MPC controller performance at constant longitudinal velocity (FPGA)

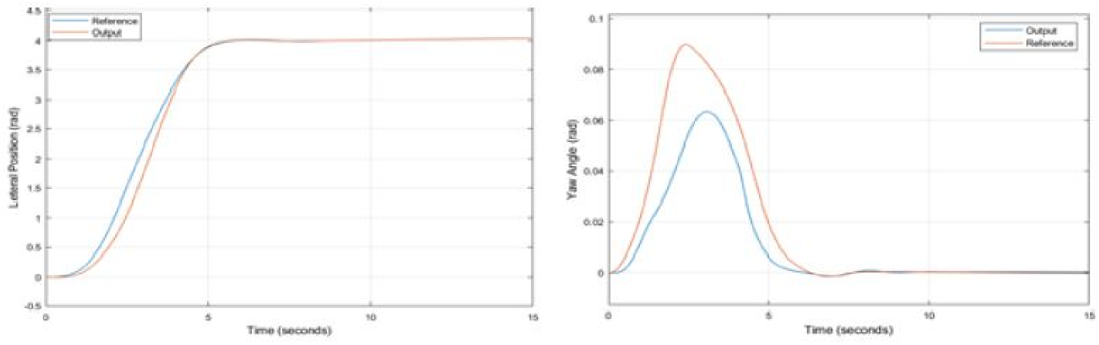

FIGURE 4.17: Adaptive MPC performance at varied longitudinal velocity (FPGA)

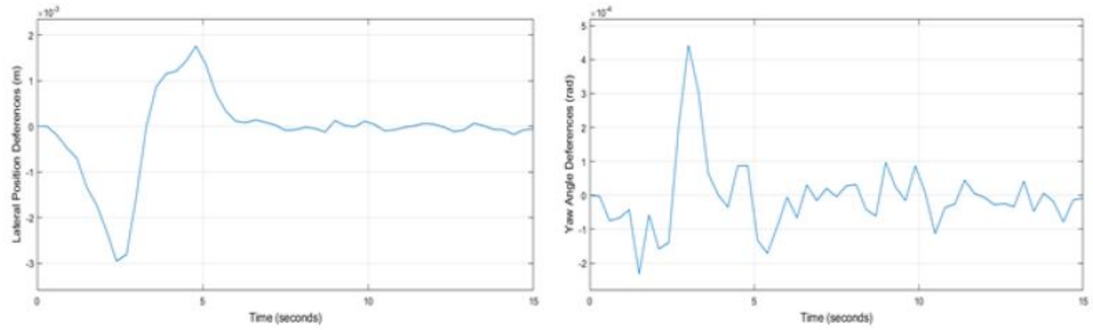

FIGURE 4.18: MPC implementation using Simulink and FPGA: Performance compression

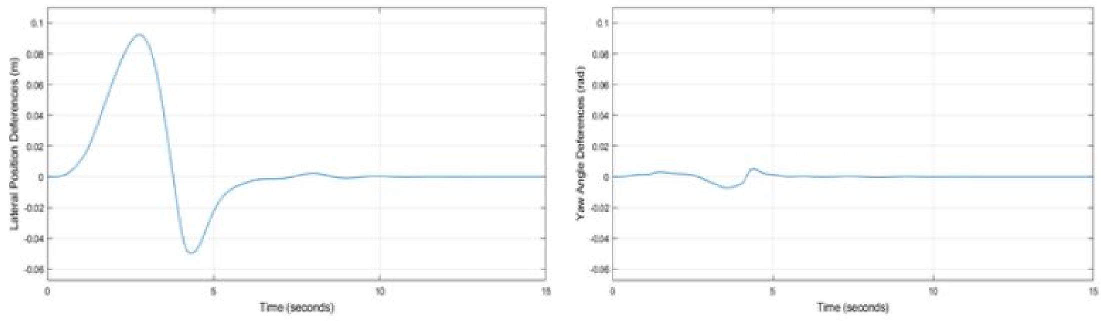

FIGURE 4.19: Adaptive MPC implementation using Simulink and FPGA: Performance compression

The implementations of MPC and Adaptive MPC controllers on FPGA were analyzed also in terms of resource utilization and power consumption using three different strategies for implementation to achieve the optimization as table 4.2 and table 4.3 show. In general, the implementations involve Logical optimization, placement of logic cells, and routing the connections between cells [Xil16]. Implementation "Defaults strategy" balances run-time with trying to achieve timing closure. The "PerformanceExplore-PostRoutePhysOpt" strategy uses multiple algorithms for optimization, placement, and routing in order to get potentially better results. In the "FlowRuntimeOptimized" strategy, each implementation step trades design performance for a better run time [Xil16]. The results in table 4.2 show that the implementation of the MPC controller on FPGA using the “defaults” strategy has the highest resource utilization, whereas the “FlowRuntimeOptimized” strategy achieved the lowest resource utilization, where the utilization of LUTs (Lookup Tables) and FF (Flip-Flop) were reduced by 13.2% and 8.86% respectively. For BUFG (Global Buffer) there is no change. On the other hand, the implementation of the MPC controller using the “PerformanceExplorePostRoutePhysOpt” strategy achieved the lowest resource utilization. Table 4.3 shows that the power consumption for all applied strategies is almost the same. Table 4.4 shows that 91% of the total power was used by the Processing System (PS), whereas only 9% was used by Programmable logic (PL) and only 6% of MMCM (Mixed-Mode Clock Manager) were used for both MPC and AMPC implementations.

Table 4.2: Resource utilization using different strategies

Defaults Strategy					
Resource	Utilization		Available	Utilization %	
	MPC	AMPC	MPC	MPC	AMPC
LUT	204	208	53200	0.38	0.39
FF	361	361	106400	0.34	0.34
BUFG	3	3	32	9.38	9.38
PerformanceExplorePostRoutePhysOpt strategy					
LUT	181	184	53200	0.34	0.35
FF	329	329	106400	0.31	0.31
BUFG	3	3	32	9.38	9.38
FlowRuntimeOptimized strategy					
LUT	177	231	53200	0.33	0.43
FF	329	361	106400	0.31	0.34
BUFG	3	3	32	9.38	9.38

Table 4.3: Power consumption – different implementation strategies

Name	Strategy	Total Power (W)	
		MPC	AMPC
Impl_1	Implementation Defaults	1.791	1.791
Impl_2	PerformanceExplorePostRoutePhysOpt	1.792	1.792
Impl_3	FlowRuntimeOptimized	1.793	1.791

Table 4.4: Power Consumption on a chip - Summary

	Power consumbtion	Power on chip	
Dynamic	91 %	Clocks	less than 1 %
		Signals	less than 1 %
		Logic	less than 1 %
		MMCM	6 %
		PS7	91 %
Static	9 %	PL Static	100 %

Thesis I

I gave a methodology for adaptive MPC development for the changing dynamics system, which significantly improved the behaviour of the controller in terms of its ability to predict and follow the dynamic changes of the system efficiently after reaching the steady state. Resulting to improve the performance and the smoothness of the driving system. To perform embedded system's deployment, I applied and analyzed different implementations of the proposed method from source utilization point of view. The implementations included logical optimization, placement of logic cells, and routing the connections between cells.

Related Publications: [RBV20], [RV20], [BRV20]

4.8 Chapter Conclusions and Summary

This chapter discussed the implementations of model predictive controller for autonomous vehicle steering task. To deal with changing dynamics systems, a linearized function was used to adapt to the new changes and provide accurate prediction at each time step (Adaptive MPC). The designed models were implemented on FPGA using MATLAB HDL coder and different strategies were adopted to optimize resource utilization. The results showed that the MPC controller provides satisfactory performance in the case of a constant dynamics system, while it couldn't handle operating conditions in the case of changing dynamics. On the other hand, adaptive MPC controller handled the changing dynamics efficiently. Additionally, the suggested model was deployed on FPGA, where different strategies were implemented to achieve the OPTIMAL behavior, including logical optimization, placement of logic cells, and routing the connections between cells.

Chapter 5

Deep Learning-Based Control Strategy for Automated Driving and FPGA Deployments Using a Novel Automatic IP Generator Tool

With the increase in the non-linearity and complexity of the driving system's environment, developing and optimizing related applications is becoming more crucial and remains an open challenge for researchers and automotive companies alike. Model predictive control is a well-known control strategy that is used for automated steering task due to its ability to solve an online quadratic optimization problem in the real-time, in addition to its efficiency in handling the environment's constraints. MPC controller drives the vehicle autonomously along the center-line of the road based on two main factors, the lateral deviation and relative yaw angle. Recently, machine learning has become an effective alternative to classical control systems, and deep learning technology has been widely used because of the promising performance achieved in different applications and tasks. In this context, I suggested that the implementation of the Deep Neural Network can achieve a great improvement in terms of efficiency compared to solving an online quadratic problem, which in turns will naturally lead to reduce the time and the complexity of implementations. This chapter can be structured in two main parts, in the first one, A developed deep neural network (DNN)-based control strategy for automated steering is provided, where the DNN model is designed and trained based on the behavior of the traditional MPC controller, and the efficiency of replacing the MPC-based controller with the suggested DNN-based model is studied and analyzed. The performance of the DNN model is evaluated compared to the performance of the designed MPC which already proved its merit in automated driving task. Taking into considerations the critical role of hardware implementations of deep neural network, a new automatic intellectual property generator based on the Xilinx system generator has been developed in order to deploy and optimize the implementations of the DNN based models on Field Programmable Gate Array. The performance was evaluated based on the ability of the controllers to drive the lateral deviation and yaw angle of the vehicle to be as close as possible to zero. The DNN model was implemented on FPGA using two different data types, fixed-point and floating-point, in order to evaluate the efficiency in the terms of performance and resource consumption. The obtained results show that the suggested DNN model provided a satisfactory performance and successfully imitated the behavior of the traditional MPC with a very small root mean square error (RMSE = 0.011228 rad). Additionally, the results show that the deployments using fixed-point data greatly reduced resource consumption compared to the floating-point data type while maintaining satisfactory performance and meeting the safety conditions.

5.1 Introduction and Background

The rapid increase in the number of cars on the roads has increased the risks associated with safety and traffic congestion. Recently, autonomous vehicles are being considered as a potential solution to overcome these problems. The autonomous vehicle can achieve more robust systems by providing more efficient driving systems and automated control [YLCT20]. Automated steering task is considered as a critical task and different control strategies can be applied such as geometric and kinematic controller, dynamic controller, optimal controller, adaptive and intelligent controller, classical controller and model-based controller [Sni09],[AZHK17]. The improvement of the applied control strategies becomes an ultimate goal for the researchers and the automotive companies which have expanded their researches and developed the used technologies. Because of that, the competition surely has intensified in this field [PLH15]. The increase in the complexity of the system is accompanied by an increase in the computational loads, consequently, an increase in the required computational capabilities which may be limited in a variety of environments [KTGL18]. In this context, and as I mentioned earlier, the MPC controller solves a constrained online – optimization problems with high computationally demanding (it is more challenging with the complex systems). Meaning that, the MPC controllers are suitable for the systems with high computing capacity environments, and it does not meet the real –time computing requirements for the resources limited environments. In addition to the computational requirements, many tasks such as those for the vehicles have real–time requirements which are directly related to the safety aspects. These requirements raised the challenges of running these applications on the different hardware computing platforms which are in turn constrained with several factors such as size, weight, cost and power consumption limitations. Consequently, the necessity of providing high efficient control strategies for the autonomous vehicle systems with minimum computational cost becomes an urgent for the practical and the economic aspects [YGDWS13], [EDC14], [SEZEHE20]. The significant developments in Machine Learning technologies including deep neural network have enabled the developers to design and develop more robust and safe autonomous vehicles systems in the complex dynamic environments. Many companies nowadays are effectively working to develop their technologies toward fully automation vehicles. The promising results of the deep neural networks associated with environment perception ,motion planning, objects detection, and traffic routing sparked interest in developing deep neural network-based control strategies [PLH15], [Sni09],[KLKK20], [nHTK15]. For example, Tesla provided the "Model S" autonomous vehicle which used DNN to develop a vision-based system for the obstacles detection task. Deep neural network is considered a self-optimized method due to its ability to optimize its behavior based on the provided information, and that makes DNNs suitable for complex dynamic systems [RHS+17]. In addition to the performance, deep neural networks offer many other benefits such as reducing the execution time and the which makes the implementations on limited-resource HW (Hardware) more efficient [LXL17], [BMKY18], It is expected to use more DNN – based systems in the near future for autonomous vehicle applications.

Custom IP generation is a tricky step but crucial for FPGA implementations. Various tools are provided for the designers to perform such steps, but all seem to be time-consuming, especially for applications that consume huge computational resources. DNNs are becoming widely used in all fields, hence, their deployment must be simplified for developers, engineers, and scientists. However, DNNs are time and resource consuming if they are implemented on sequential computational systems such as $\mu P/\mu C$ (microprocessors/microcontrollers) or Digital Signal Processors (DSPs) [QZF+20]. That is why DNNs are more likely to be implemented with parallel computing systems such as FPGAs, field programmable analog arrays (FPAAs), graphical processing units (GPUs) or dedicated hardware accelerators on application-specific integrated circuits (ASICs). Some papers propose implementations of model predictive controllers on FPGAs considering different implementation methods such as high-level synthesis [KLKK20], a Xilinx System Generator [STH20] or even HDL [nHTK15].

Generally speaking, it is evident that the most economical solution for these applications is to adapt a dedicated ASIC conditionally upon mass production. GPU are known for their ability to execute several parallel processes, which makes their application for image-processing favorable [CDKDD21]. Since the neural networks can be computed in a parallel way, GPU can be dedicated for such applications [MBBH21], [SH17]. However, they are known to be power consuming, which makes their use inadequate for embedded applications. Re-configurable computing is an efficient alternative solution for parallel processes where all the computations can be executed at the same time. The most common re-configurable technologies are FPGAs [WGDL21], in addition, recently FPAAs have become a hot topic for research [MBBH21], [SH17]. However, because of their lack of hardware resources, the use of FPAAs are limited to scientific research applications [DM20]. Many studies have proven that FPGA-based neural network implementations provide much better results in terms of power consumption and timing performances [CDKDD21], [LHW20]. the lack of FPGA hardware resources restricts their use to limited DNN sizes, hence, an FPGA-GPU heterogeneous platform could be a solution [CHPB21].

Despite the increasing popularity of Python, MATLAB is still a powerful environment that users in various fields are utilizing for a huge range of applications. A MATLAB toolbox was released in September 2020 for the creation of DNN IPs for FPGAs [Mat18b], but this tool forces the developer to use high-end FPGA, even for small DNN structures. Some works propose different methods or tools to implement DNNs on FPGA [GS20], [GLX^{+}17].

The suggested solution is to develop a deep neural network model based on the behavior of the traditional MPC controller so that the DNN model can replace the MPC controller in high complexity driving system environments. Additionally, one of the motivations behind the work is to propose an alternative tool to implement deep neural networks on low-end FPGA. The work includes creating the back-end of an automated tool to generate deep neural network Intellectual Properties for further FPGA implementation. The tool is based on the Xilinx System Generator, where blocks are automatically invoked, parameterized, and linked from the script. The targeted FPGA which is used in this work is the *Xilinx Kintex-7- KC705* chip, which is known for its hardware resources limitations. Nevertheless, the new tool has the ability to deploy the generated IP on larger and smaller FPGAs/SoCs.

Autonomous vehicle systems mainly based on the data acquired from the sensors in order to provide an accurate perception of the surrounding environment. The role-based controllers are the conventional controllers used to control the early autonomous vehicles where their parameters are tuned by the developers during the simulation process. Because of the non-linearity nature of the driving systems and the complex environments, these controllers are not efficient enough [SBS13]. Deep neural networks have become an alternative to the classical approaches and has gained a great attention as they have shown considerable performance and overcomes the problems of these approaches.

5.2 Design of the MPC Controller

The MPC drives the vehicle to the target point along the desired trajectory by controlling the lateral deviation d and the relative yaw angle θ of the vehicle. Maintaining these variables to be zero or as close as possible to zero is the online optimization problem that the MPC controller must handle in real time. Since MPC is a model-based controller, the design process has two main steps, designing the plant model (the vehicle) first and then designing the MPC controller in the second step. The design process includes tuning the parameters of the controller and formulating the operating conditions that are imposed by the system in the form of soft and hard constraints.

5.2.1 The vehicle Model

The dynamic model is represented by equations 5.1, 5.2 and 5.3. Figure 5.1 shows the global position of the vehicle, where v_x, v_y are the longitudinal and the lateral velocities respectively, d is the lateral deviation, m is the total mass of the vehicle, l_r is the distance between the rear tire and the center of the gravity, l_f is the distance between the front tire and the center of the gravity, l_z is yaw moment, c_f, c_r are the corner stiffness of the front and rear tires respectively, δ is the steering angle, θ is the yaw angle, ρ is the curvature, and ω is the yaw rate. The lateral and yaw motions are determined by the fundamental laws of motion, meaning that they are determined by the forces that are applied on the front and rear tires.

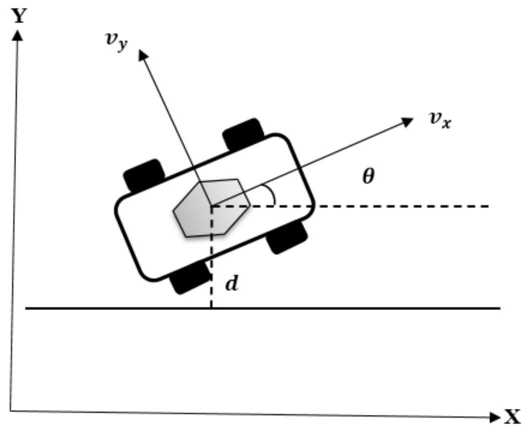

FIGURE 5.1: Global position of the vehicle model.

$$\frac{d}{dt}\begin{bmatrix} v_y \\ \omega \\ d \\ \theta \end{bmatrix} = \begin{bmatrix} A \end{bmatrix} \begin{bmatrix} v_y \\ \omega \\ d \\ \theta \end{bmatrix} + \begin{bmatrix} B \end{bmatrix} \begin{bmatrix} \delta \\ \rho v_x \end{bmatrix} \tag{5.1}$$

$$A = \begin{bmatrix} -\frac{2c_f+2C_r}{mv_x} & -v_x - \frac{2c_f l_f - 2c_r l_r}{mv_x} & 0 & 0 \\ -\frac{2c_f l_f - 2c_r l_r}{I_z v_x} & -\frac{2C_f l_f^2 - 2C_r l_r^2}{l_z v_x} & 0 & 0 \\ 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.2}$$

$$B = \begin{bmatrix} \frac{2c_f}{m} & 0 \\ \frac{2c_f l_f}{I_z} & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.3}$$

5.2.2 The MPC model

The first step in designing the MPC model is to determine the input-output signals of the vehicle model and the second step is to set the parameters and determine the constraints. The manipulated variable (steering angle δ) and the disturbance (v_x ρ) are determined as inputs, while lateral velocity v_y, lateral deviation d, yaw angle θ, and yaw rate ω are determined as outputs as figure 5.2 is shown. The design parameters of the MPC controller were tuned during the design process and the standard recommendations were taken into consideration to determine their values. Sample time (T_s) determine the rate that MPC controller executes the control algorithm. In the case of long T_s, the controller will not be able to response in time to the disturbance. On the other hand, in case of too small T_s, the controller will response faster, but the computational loads will increase. Prediction horizon (P) is chosen in a way that covers the dynamic changes of the environment. The recommendation is to chose P to be 20 to 30 samples. By taking into consideration that the first two control actions have the highest impact on the response behaviour, determining a large control horizon (M) increases the computational load, while a small M increase the stability. The parameters of the MPC model are determined as flows: the sample time $T_s = 0.1\ s$, the prediction horizon $P = 3\ s$, and the control horizon $M = 2\ s$. The constraints are determined as follows: the steering angle is in the range [-1.04, 1.04] rad and the yaw angle rate is in the range [-0.26, 0.26] rad. The parameters were maintained during the design process until satisfactory behavior was obtained. The overall design of the MPC and plant model is shown in figure 5.3.

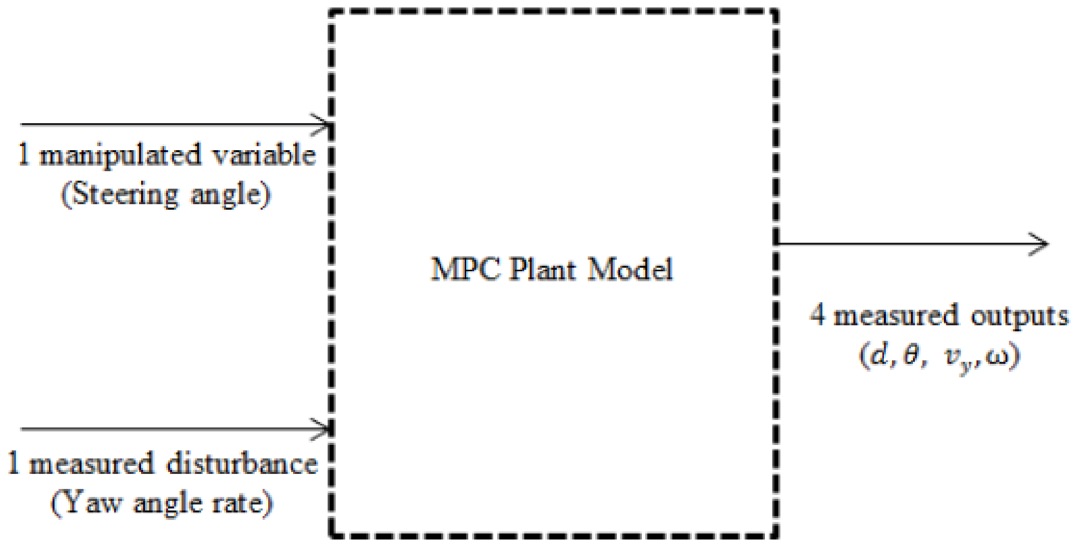

FIGURE 5.2: Input/Output signals of the MPC plant model.

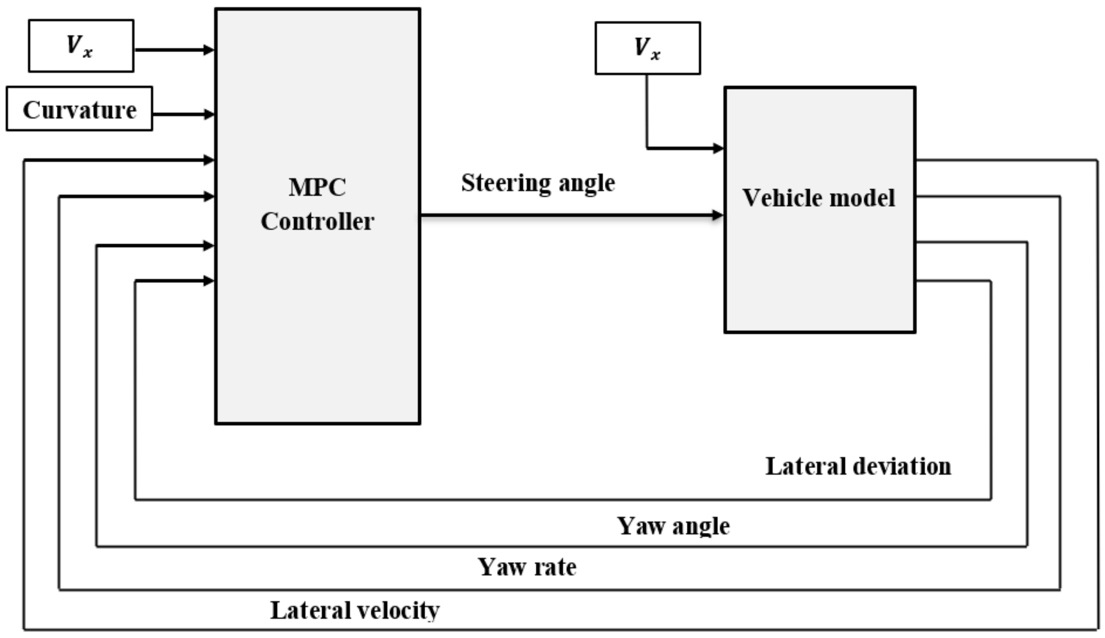

FIGURE 5.3: General MPC and plant model diagram.

5.3 Design of the Depp Neural Network Model

Designing the deep neural model goes through several steps including designing the model architecture, defining the training options, and preparing the training data. The DNN model is developed using Matlab environment.

5.3.1 Design of DNN Model Architecture

In this step, the layers of the deep neural network architecture are chosen, and these layers are defined based on the nature of the task as each layer has a specific function (See figure 5.4) . The suggested architecture of the DNN controller model consists of 8 layers which are the input layer, 6 fully connected layers (FC) and each FC layer has activation function (ReLU). The last layer is a regression layer which holds the loss function (mean-squared-error).

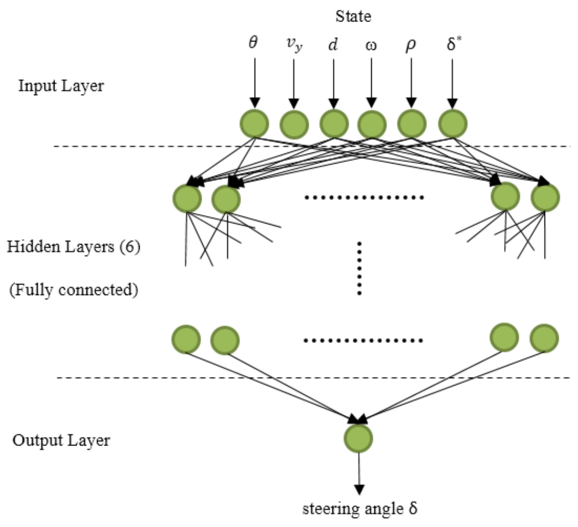

FIGURE 5.4: The DNN model architecture representation.

5.3.2 Training Options and Data Preparation

In this step, the optimizer (solver) for the suggested model is defined, where there are different types of optimizers based on the problem for which the model will be implemented. The optimizer can be specified as one of three types, Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (ADAM) or Root Mean Square Propagation (RMSProp). In addition to the optimizer, a set of options are also determined. Table 5.1 shows the main training options that are chosen to train the model. The data set is generated by implementing the MPC controller against a massive number of scenarios that cover the maximum number of the possible environment's states, and then obtaining and recording the control actions in the data set. The size and type of the generated data set is (120000 x 6), double data type, where 6 refer to the number of the state space variables. The generated data set is divided into three sets, which are: training set that is used to train the model, validation set that is used to validate the model during the training and testing set that is used to test the model after being trained. After designing the deep neural network, defining the training options, the training process is performed using the training and the validation data sets. The training stops after the final iteration.

The details showed that 9680 iterations are needed to perform the training, 40 epochs and 242 iterations per each (40 *242 = 9680). The validation process was performed every 50 iterations. The validation loss (root mean square error RMSE) was almost the same for each mini batch (RMSE = 0.010799), which means that the trained DNN does not over fit. After, the trained neural network was tested using the testing data set. The performance of the DNN model is evaluated comparing to the performance of the MPC controller, where the RMSE between the outputs of the controllers is calculated. The obtained root mean square error by the end of the testing process was: RMSE = 0.011228, which is a very small compared to the range of the steering angle [-60 – 60]°. This small value indicates to that the DNN model successfully imitated the behavior of the MPC controller.

TABLE 5.1: Training options of the DNN model.

Training option name	Value	Description
Solver	ADAM	The optimizer
MaxEpoch	40	Maximum number of Epoch
MiniBatchSize	420	Size of mini batch that is used for each iteration
InitialLearnRate	0.001	Initial learning rate
LearningRateSchedule	Piecewise	The learning rate is updated every specific number of epochs
LearningRateDropPeriod	20	The learning rate is updated every 20 epochs
LearningRateDropFactor	0.1	The factor that is used to update the learning rate

5.4 Auto Generation of DNN IP Procedures

The procedure of the neural network's IP auto-generating has two main steps, as shown in figure 5.5. First, the user is asked to define the parameters concerning the DNN (the structure, the data type and the activation function) and the targeted computational HW and the values of weights and biases parameters are imported from the pre-trained DNN. Then comes the step of setting the input/output interfacing mode, where 4 ways are available: UART, AXI, constrained parallel, and no interface modes. If the UART interfacing mode is selected, two additional pre-designed IPs are invoked that are responsible for receiving and transmitting UART data from/to the DNN IP. In this mode, the user is asked to specify the ports to be used for Tx and Rx. If the AXI mode is selected, it provides the possibility for the DNN IP to communicate with the processing system or the soft microprocessor core. If UART or AXI modes are set, additional IPs will be invoked (UART_Tx IP, UART_Rx IP, AXI-interface IP, and Processor System

Reset IP), hence, some additional hardware resources and power are required. The other disadvantage of the AXI interfacing mode is its power consumption since PS consumes 1.53 W. In our work, only the AXI interfacing mode is utilized (the UART mode will be treated in a future work). The third interfacing mode is the constrained parallel I/O where the user is asked to specify the ports to be used. This mode allows parallel communication from/to DNN, and hence there will be no latency caused by the data transmission and reception, however, this method consumes a lot of I/O resources. It is therefore not practical for the majority of the DNN applications. If no interface mode is selected, the connection is unconstrained, which permits IP-IP interconnection. Table 5.2 summarizes the interfacing possibilities for the presented tool, where AXI interface remains the appropriate one for the studied application in our work. After setting up the DNN-IP preferences, the XSG automation part begins, which consists of invoking the elementary computational components needed for each neuron, linking the components and the neurons, setting the weights and biases accordingly, implementing the I/O, and then generating the IP. Figure 5.6 shows the auto-generated DNN circuit on the Xilinx System Generator to be implemented on a low-end FPGA.

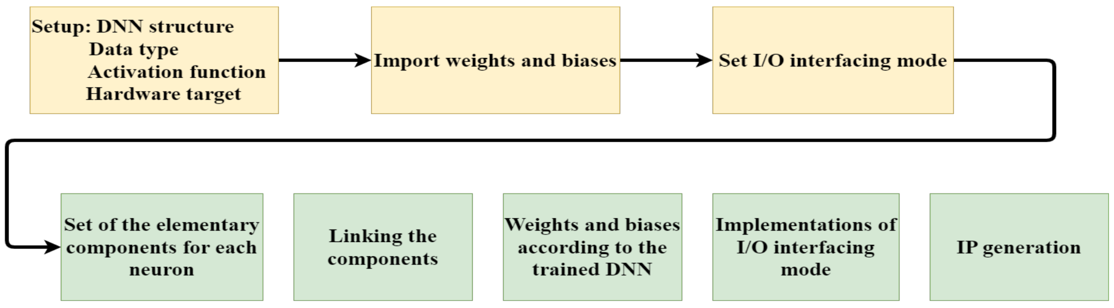
Setup: DNN structure
Data type
Activation function
Hardware target
Import weights and biases
Set I/O interfacing mode
Set of the elementary components for each neuron
Linking the components
Weights and biases according to the trained DNN
Implementations of I/O interfacing mode
IP generation

FIGURE 5.5: Flow of deep neural network IP automated generating.

TABLE 5.2: DNN IP generator tool interfaces

	Connectivity	**Speed**	**Power**	**HW resources**
Constrained I/O	PL-out	Very high	Very low	Null
Unconstrained I/O	IP-IP	Very high	Very low	Very high I/O
UART	PL-out	Low	Low	Low
AXI	PL-PS	High	High	High

5.5 Implementation of the Solution Considering the DNN on a HW accelerator

The implementation of the FPGA design is not a straightforward process due to the lack of a direct connection between the algorithms' design and the hardware, in addition to

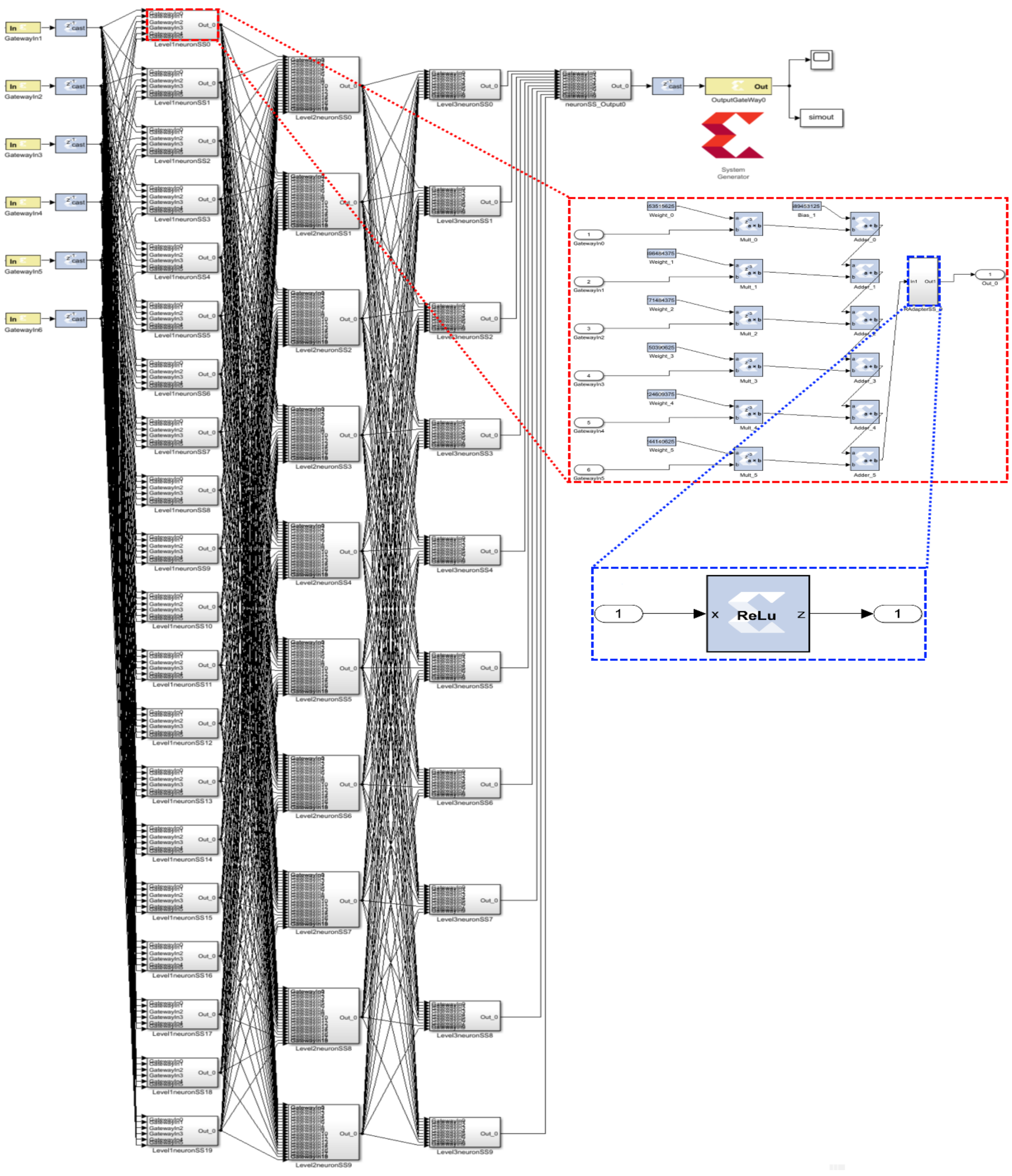

FIGURE 5.6: The auto-generated deep neural network structure on Xilinx System Generator to be implemented on a low-end FPGA.

the deviations that can be caused by the difference between the fixed-point and floating-point implementations of the algorithm's specifications. Also, the hand-written code is error-prone and can be hard to debug. In order to address these problems and provide an integrated workflow with an unified environment for algorithm design, simulation, validation, and implementation, the suggested solution was performed using Matlab, Simulink, and the Xilinx System Generator-based tool "Automatic DNN IP Generator". Figure 5.7 shows the detailed steps of implementation.

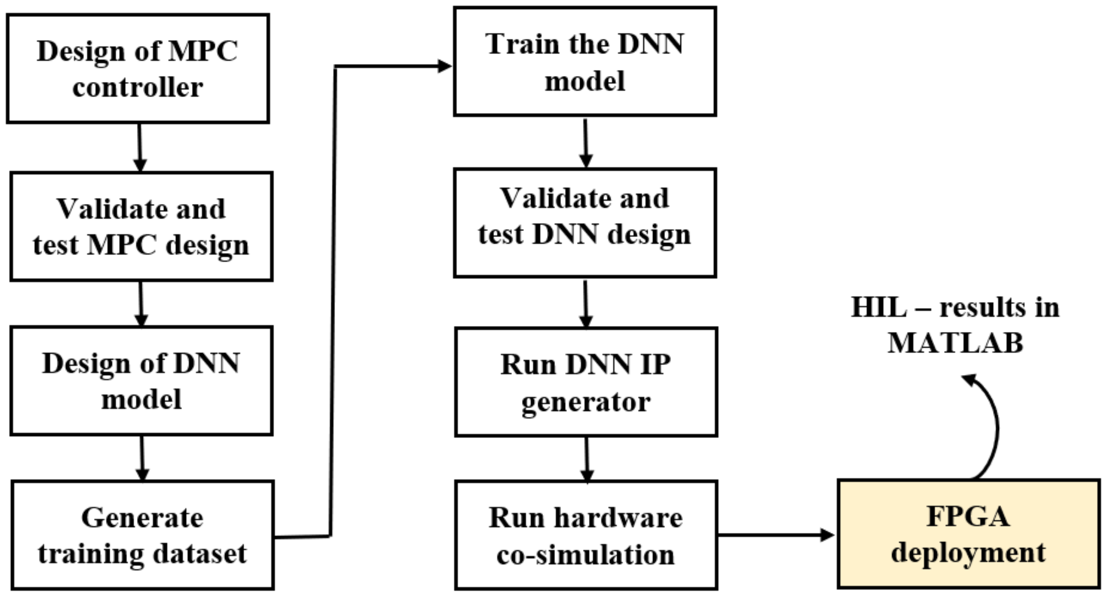

FIGURE 5.7: The implementation steps of the solution.

5.6 Results and Discussion

The implementations of the MPC, the DNN model, and the deployment of the DNN on FPGA are discussed taking the response of the MPC as reference behavior. In addition to the performance, the deployments using floating point and fixed-point data type are discussed in terms of resource consumption. The performance of the controllers are evaluated based on the settling time T_s, the overshoot M_p, and the final value (steady-state) of the yaw angle and the lateral deviation of the vehicle. The overshooting shows the amount that the lateral deviation/yaw angle overshoots its target value, while the settling time shows the time required to settle and reach the final value within a certain percentage. The settling time of the performance indicators were determined to be the time that the signal reach 5% (commonly used) of its final value. Figure 5.8 clearly shows that the DNN model and the traditional MPC have a very similar response, meaning that the DNN model successfully imitated the behavior of the MPC, while the variant of DNN on FPGA has a slightly different response. In order to evaluate these behaviors, figure 5.9, figure 5.10 and table 5.3 present the performance indicators, which show that the MPC, the DNN model, and the DNN on FPGA, all successfully drive the lateral deviation and yaw angle to be zero or very close to zero as a desired steady state. The detailed results show that the settling time for both indicators is almost the same in the case of the traditional MPC and the DNN model, while approximately 0.342 s in the case of lateral deviation and 0.3013 s in regards of yaw angle were noticed in the behavior of the DNN on FPGA version. On the other hand, the behavior was very similar in regards of the overshooting, where only 0.0492 m for the lateral deviation and 0.0339 rad for the yaw angle were noticed in the response of the DNN on FPGA compared to the response of the traditional MPC. These results demonstrate that the trained DNN model provided satisfactory performance and the vehicle was driven smoothly to the desired destination. Despite the slightly higher overshooting and settling time, the DNN model after being deployed on FPGA provided a satisfactory performance and meets the safety requirements that were determined in the designing process. Additionally, the implementations proved that our DNN model provide an improvement in term of execution time, where it provided a response 4.5 times faster compared to the traditional MPC controller as it is shown in table 5.4. In addition to performance, in order to evaluate the efficiency of deploying the DNN on FPGA in terms of resource consumption, the main estimated resource utilization of the deployments using fixed-point and floating-point data types were compared and presented in table 5.5. The results show that DNN on FPGA using fixed-point data consumes fewer resources compared to using floating-point data, where 86.29% of the LUTs and 51.54% of the DSPs were saved from the overall resource availability of the FPGA board.

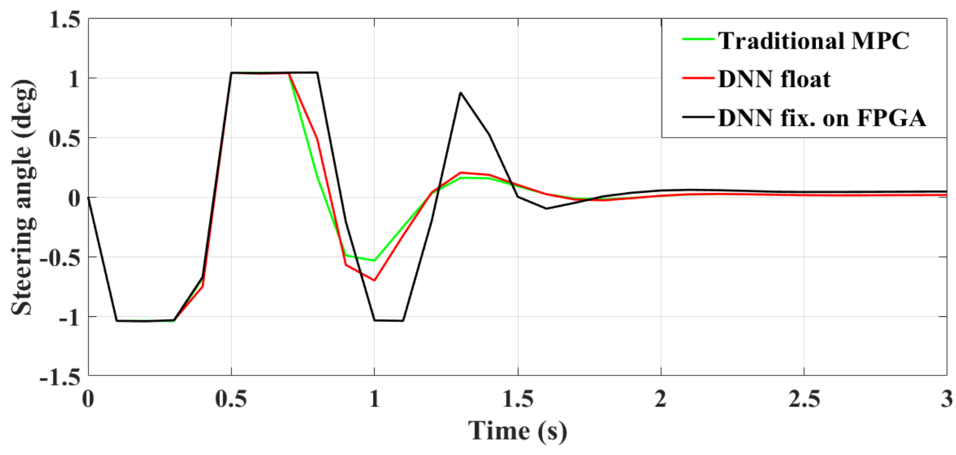

FIGURE 5.8: Estimated steering angle computed on FPGA compared to simulation results of the traditional MPC and the DNN model.

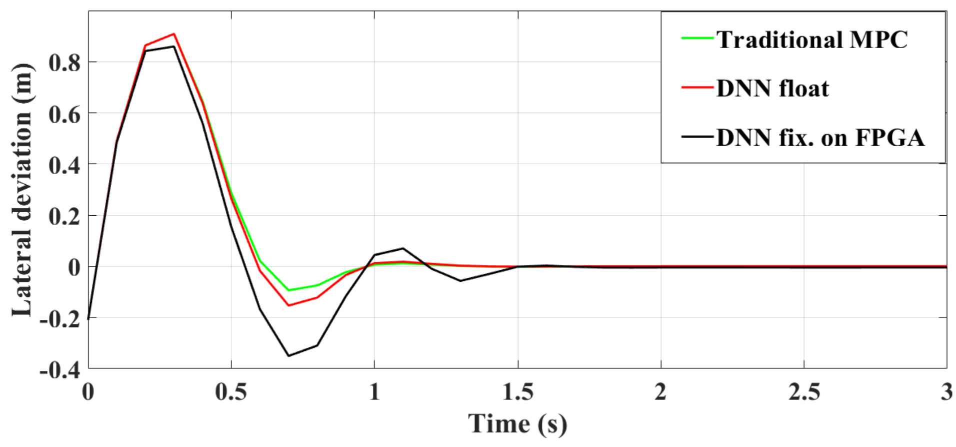

FIGURE 5.9: Vehicle response in terms of lateral deviation: comparison of FPGA implementation and simulation results.

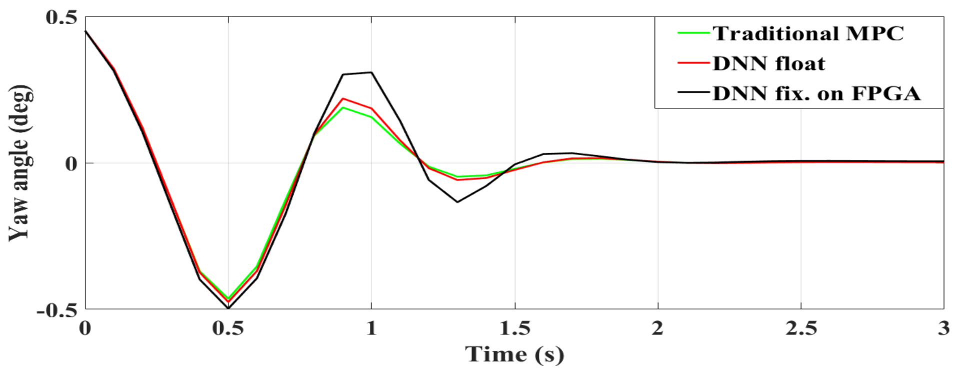

FIGURE 5.10: Vehicle response in terms of yaw angle: comparison of FPGA implementation and simulation results.

TABLE 5.3: Performance indicators for controller behavior

Performance Indicator		Lateral Deviation	Yaw Angle
Traditional MPC	Overshooting	0.9086	-0.4637
	Settling Time	1.125	1.4912
	Final Value	0.00001	0.00166
DNN	Overshooting	0.9086	-0.4749
	Settling Time	1.1924	1.5067
	Final Value	0.00063	0.00169
DNN on FPGA	Overshooting	0.8592	-0.4976
	Settling Time	1.4674	1.7925
	Final Value	-0.00427	0.00504

TABLE 5.4: Execution time comparison

Controller	Execution Time (s)
MPC controller	1.54
DNN controller	0.34

TABLE 5.5: Resource utilization of DNN on FPGA for fixed point versus floating point data

Site Type	Used		Available	Resource Utilisation [%]	
	Fixed-Point	Floating-Point		Fixed-Point	Floating-Point
LUTs	7166	183034	203800	3.52	89.81
DSPs	393	826	840	46.79	98.33

Thesis II

I proposed deep neural network-based control strategy as an alternative solution to the classical MPC controller for automated driving task, aiming to reduce the complexity of solving the online-optimization problem, therefore the execution time. I designed and trained the DNN-based controller to imitate the behaviour of the traditional MPC controller. I also proposed a new automatic intellectual property generator tool, which is developed not only to perform but also to optimize the deployments of deep neural networks on low-end Field FPGA. The results shows that DNN-based model successfully *imitated the behaviour of the classical MPC and reduced the exaction time. The new Automatic DNN IP Generator proved its efficiency to perform FPGA deployment.*

Related Publications: [RBV21], [KRVB20a], [ch5]

5.7 Chapter Conclusions and Summary

This chapter concerns the inefficiency of the classical MPC controller with the complex automated driving environment. A deep neural network model-based model was suggested as an efficient alternative to the classical MPC controller. The DNN model was trained to imitate the behaviour of the MPC controller. The designing, testing and validation processes were discussed in details. Also, this chapter concerns the deployment of the DNN model on low end FPGAs, where a new tool based on the Xilinx System Generator was developed to perform and optimize the deployments of the DNN model on FPGAs. The obtained results show that the suggested DNN model efficiently succeeded in emulating the behaviour of the MPC controller and reduced the execution time by 3 to 4 times. As a result, the traditional model predictive controller can be replaced efficiently by the suggested DNN model, which in turn reduces the execution time. On the other hand, the trained DNN model was efficiently deployed on low-end FPGA *Xilinx Kintex-7 FPGA KC705* using floating and fixed-point data type, achieving satisfactory performance and meeting the design's constraints.

Chapter 6

A Hybrid Machine Learning-Based Control Strategy for Automated Driving Optimization

Developing autonomous vehicles is a highly important topic in the field of intelligent transportation systems. Automated steering is a crucial function in autonomous vehicles. Therefore, it is urgent to either develop a new effective control strategy or improve existing ones. A variety of control strategies are used for this purpose, most with limitations related to their computing capabilities with the highly complex systems or to lack of efficacy related to maintaining the balance between driving performance and driving smoothness. This chapter concerns the use of machine-learning algorithms for automated driving, where a new method is introduced to achieve performance optimization. The new method leverages the advantages of supervised learning and reinforcement learning algorithms in one control model to achieve a better generalization capability within the complex driving environment. The efficiency of the suggested model is analyzed and compared to two other machine learning methods. To sum up, in this chapter, three different machine learning-based models were developed to perform an autonomous driving task: a supervised learning model (Deep Neural Network), a reinforcement Deep Q-learning model (DQN), and a hybrid model. The DQN was designed with the same structure as the DNN and trained by directly interacting with the driving environment. The hybrid model is a combination of supervised and reinforcement learning algorithms, where the trained DNN model is used as a decision-maker (Actor) in a Deep Deterministic Policy Gradient(DDPG) reinforcement learning model. The behaviors of the designed models were compared based on several performance indicators, including the ability to drive the vehicle along the desired trajectory, the response time, and the smoothness of the driving system.

6.1 Introduction

The evolution of autonomous driving systems has seen the use of different technologies aiming to improve efficiency, enhance driving safety and reduce the risks related to traffic congestion. Driving in a structured environment and highway driving projects were some of the earliest autonomous vehicle projects, carried out at Carnegie Mellon University and Bundeswehr University Munich [DZ87]. Since then, projects and research related to autonomous vehicles have been carried out by academic institutes and companies alike.The vehicle interacts with the surrounding environment in order to perform several related tasks: perception, where the required information about the driving environment is provided to the system; planning, where the optimal scenarios and the control actions are obtained based on the provided information; and the control function, where the control strategy is put into action [BSBG12]. The automated steering task is a part of the control function, where the tracking errors are minimized in order to follow the desired trajectory. Driving the vehicle along the desired trajectory is considered one of the most critical tasks due to the fact that any failure in the applied control strategy can have severe consequences. A variety of control strategies have been used to perform the automated steering task, such as the classical feedback control algorithm, Model-Based Control, Dynamic Control, and Adaptive Control [HGCV21], [LWZC15], [WLTY08]. In this context, Model Predictive Control has become the most commonly used algorithm for the autonomous vehicle steering system. The MPC controller solves an online optimization problem with the ability to handle the system constraints (soft-hard) by including them in the design process, which makes it a powerful strategy to deal with the stability and the changing dynamics of the vehicle. On the other hand, with the increase of the system complexity, the computational load of the MPC controller is increased, since it solves the optimization problem in each time step, and it may not be able to meet the real-time requirements. Additionally, MPC is resource-consuming, which makes it invisible, especially when it comes to the limited resources of embedded computing platforms such as system-on-chip and field-programmable gate array adaptive platforms [BFEG17]. Recently, Deep Neural Network has gained attention and has been rapidly developed and efficiently implemented with a variety of applications in different fields such as image classification [KSH17], natural language processing, and speech recognition [DLH+13]. In contrast to the classical control algorithms which are mainly based on tuning predefined parameters related to a determined environment [KBJ+20], the behavior of the deep neural network model is optimized based on the provided information (self-optimized algorithm). In other words, the neural networks algorithm bypasses the need for significant parameter tuning, which makes it more efficient to model highly complex systems and to deal with unforeseen situations, especially after being well trained and validated using sufficient data-sets. Recently the implementation of deep neural networks within the domain of robotic applications has made massive progress and has provided promising results such as perception and motion planning [PSN+17] and object detection and semantic segmentation [GDDM14b]. In contrast to supervised learning, agents in Reinforcement Learning (RL) are trained by directly interacting with their environment rather than explicitly guiding the model on how to act based on the labeled data [Mah20]. The performance of the RL agent is evaluated based on the reward function, where the agent is trained to act in the environment in a way that maximizes the cumulative reward in order to improve the performance [LLL+19]. RL has proven to be a powerful method mainly in the domains of game playing and robotic manipulation [GLL17], and RL algorithms are considered a promising potential solution for many other applications, especially in cases where classical supervised learning is not applicable. Although there are promising results achieved by the implementations of reinforcement learning with different complex tasks related to automated driving, RL is still an emergent field in this domain, where the implementations and deployment of real-world applications are still very much an open challenge and RL has not yet been applied to practice as successfully as supervised and unsupervised learning.

The main contributions of this work can be summarized in two main points. The first is leveraging the advantages of reinforcement learning and supervised learning by combining them in one control model in such a way that the reinforcement learning-based model optimizes the actions that are taken by the supervised neural work (DNN) model. The second contribution comes in enriching the research on RL algorithms and paving the way to bring RL closer to real-world implementations. In chapter 4, a classic MPC controller was designed and deployed on FPGA for automated driving task, while in this work three different machine learning-based models are developed for the same task and compared to the traditional MPC.

The first model is a DNN-based model which is designed and trained using a supervised dataset.

The second model is a reinforcement learning-based model (DQN) which is designed and trained without any supervision data, but directly by interaction with the environment.

The third model is a hybrid one, which is a combination between the DNN and reinforcement learning methods. The trained DNN will be used as decision maker working beside another network (critic) within a DDPG reinforcement model.

The combined method is expected to provide an optimized solution, as the actions that are taken by the decision maker (trained DNN) will be evaluated and optimized by another neural network in order to minimize errors. Additionally, the combined model will be able to deal with and adapt to new cases that have not been faced during training.

6.2 Background

In this section, an overview of the reinforcement learning algorithms and the most relevant studies are presented. Also, the RL algorithms were briefly compared to the supervised algorithms, presenting the key differences.

6.2.1 Reinforcement Learning Algorithms and Related Works

Sequential decision-making problems can be formulated by Markov Decision Processes (MDPs), which are considered as a bedrock of the problems that reinforcement learning solves. MDPs consist of a decision maker (agent), set of states (S), set of actions (T), transaction function (A), and reward function (R) which can be represented as a tuple <S, A, T, R>. At each time step (t), and based on the received state ($S_t \in S$), the agent takes an action ($A_t \in A$) which represents a pair (A_t, S_t) in the next time step. Based on the taken action the environment is transitioned to a new $S_{t+1} \in S$, and the agent receives a reward $R_{t+1} \in R$, [Put14] (see figure 6.1). The cumulative reward is simply represented as a sum of the expected return at each time step and can be mathematically formalized as it is shown in equation 6.1. In the case of tasks that have a final time step, the interaction between the agent and the environment is performed based on episodes, where each episode has a terminal state at the final time step T. At the beginning of every next episode, the environment is reinitialized to some standard state, meaning that each episode starts running independently from the final state of the previous episode. The notion of total return at each time step becomes a problem when it comes to the continuing tasks, where the agent interacts with the environment continually, meaning that the final time step will be T$= \infty$, and here is where the discounted return comes into the picture, where the agent seeks to maximize the discounted future return which is represented in equation 6.2. The concept of discounted return makes the future return heavily discounted compared to the immediate reward and that in turn makes the actions that are taken by the agent more influenced by the immediate return.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{6.1}$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^3 R_{t+2} + ... = \Sigma_{i=0}^{\infty} \gamma^{i} R_{t+i+1} \tag{6.2}$$

Where G_t is the total reward at the time (t), $(R_{t+1}, R_{t+2}, ..)$ are the rewards of the next steps, R_T is the reward of the final time step, and γ is a discounted factor and it is designed to be in the range $[0 - 1]$. In the case of a small value of γ, the agent will seek to maximize the short-term reward, in contrast, the high value will encourage the agent to look more forward and seeks to maximize the rewards of the longer term. Generally speaking, the discounted factor is chosen to be closer to the value 1 in the case of the finite number of time steps of the MDP model, or goal-oriented applications in order to force the agent to focus on the goal. Whereas, in the case of infinite time steps, the discounted factor is chosen to be smaller in order to make the balance between the long- and short-term rewards. The probability of selecting an action by the agent from all possible actions at all possible states is determined by the policy (π) that the agent follows. In addition to the probability of the selection action, the value function evaluates how good it is for the agent to select an action at a given state under a policy, and this is called the action-value function (q_π), or how good it is for the agent to be at a given state following a policy, and this is called the state-value function (v_π). Equations 6.3 and 6.4 are the mathematical representations of the action-value and the state-value functions, respectively.

The action-value function $q_\pi(s, a)$ is the expected reward ($\Sigma_{k=1}^{\infty}\gamma^k R_{t+k+1}$) starting from the state (s) at the time (t), performing the action (a) and following the policy, where the state-value function v_π (s) is the expected reward starting from state at the time (t) and following the policy. It is worth mentioning that q_π is also referred to as the Q-function and its output is called the Q-value (the quality of taking an action). In terms of optimality, the main goal of the RL algorithm is to select the optimal policy that will yield the highest expected reward for each state. The optimal policy is associated with an optimal state-value function (v_*) and an optimal action-value function (q_*) or optimal Q-function, which are represented in equations 6.5 and 6.6, respectively. The fundamental property that the optimal Q-function (q_*) must satisfy is the Bellman equation (see equation 6.7), where R_{t+1} is the expected reward that the agent obtains by taking the action at state, whereas $\gamma \max q_*(s^{'}, a^{'})$ is the maximum expected discounted reward that can be received from any next state-action pair [How60], [SB99]. Reinforcement learning is a category of machine learning that studies the behavior of an agent and focuses on how this agent might interact with its environment. The main goal of the agent is to maximize the cumulative given rewards it receives over time in order to optimize its behavior in such an environment [FLVH$^+$18]. Based on the fact that the agent is able to learn the value function estimates or/and the policies directly, RL methods can be categorized into three main methods: value-based methods, policy-based methods, and actor-critical methods [Li17]. All of the methods share the same strategy of determining the actions and evaluating the agent behavior, but the essential difference is where the optimality resides.

$$q_\pi(s, a) = E_\pi(\Sigma_{k=1}^{\infty}\gamma^{^k} R_{t+k+1} \mid S_t = s, A_t = a) \tag{6.3}$$

$$v_\pi(s) = E_\pi(\Sigma_{k=1}^{\infty}\gamma^{k} R_{t+k+1} \mid S_t = s) \tag{6.4}$$

$$v_*(s) = \max_\pi v_\pi(s) \tag{6.5}$$

$$q_*(s,a) = \max_\pi q_\pi(s) \tag{6.6}$$

$$q_*(s,a) = E_\pi(R_{t+1} + \gamma \max_{a'} q_*(s', a')) \tag{6.7}$$

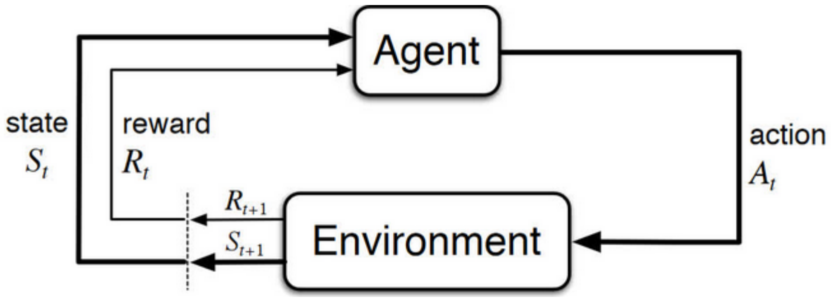

FIGURE 6.1: Markov Decision Processes.

Value-Based Algorithm

Value-based methods aim to get the optimal cumulative reward and determine the optimal policy that follows the recommendations. One of the most commonly used reinforcement learning value-based algorithms is the Q-learning method [ADBB17]. The objective of Q-learning is to find the optimal policy by learning how to find the optimal Q-value for the (s, a) pair, where the Q-values are stored in a Q-table. The Q-learning algorithm uses what is called the value iteration approach to converge the Q-function to the optimal Q-function by iteratively updating the Q-value for each (s, a) pair using the Bellman equation. In reinforcement learning, the agent may exploit its knowledge by selecting an action that is known to provide a positive reward in order to maximize the total reward. On the other hand, the agent may take the risk of exploring unknown–actions that may lead to a lower reward or discover actions that provide a higher reward compared to the current best-valued action. Managing the exploration-exploitation trade-off is considered a critical challenge in the RL algorithm and ε- greedy policy is one of the most commonly used strategies to overcome this challenge. In this strategy, the exploration rate ε refers to the probability of exploring the environment rather than exploiting it. Meaning that the agent either will choose an action randomly with probability (ε) or will be greedy by selecting the highest valued action with probability (1- ε). Since the agent at the beginning know nothing about the environment, the exploration rate (ε) usually is initialized to be 1, so the agent starts with exploring the environment. With the training progress, the agent learns more about the environment and should conduct more exploitation, which can be achieved by gradually decreasing the probability of exploration (ε) by a specific rate at the beginning of every new episode. After taking the action, the agent observes the next state, obtains the gained reward, and updates the Q-value (Q-table in general) based on the Bellman equation. The objective is to converge the Q-value to the optimal Q-value (q_*) by driving this value to be as close as possible to the right-hand side of the Bellman equation. Another factor is used to update the Q-value and it is called learning rate (α) which determines how fast the agent will adopt the new Q-value and it is in the range [0-1]. In other words, by using the learning rate, the old Q-value is not completely overwritten, but it is updated by taking into consideration how much information should be kept from the old computed Q-value. A higher learning rate will drive the agent to adopt the new Q-value faster as equation 6.8 shows.

$$q^{new}(s,a) = (1-\alpha)q(s,a) + \alpha(R_{t+1} + \gamma \max_{a^{'}} q_*(s^{'},a^{'})) \tag{6.8}$$

where $q^{new}(s,a)$ is the new value, α is the learning rate, (1-α)q(s, a) is weighted old value and $R_{t+1} + \gamma \max_{a^{'}} q_*(s^{'},a^{'})$ is the learned value.

With the increase in environment complexity, the state space size increases, and the performance of the Q-learning method will drop off because of the value iteration strategy that is used to update the Q-values (Q-table). The problem with large MDPs is that there are too many states and/or actions to be stored in the memory, and it is too slow to calculate the value for every individual state [LHP+15]. To overcome this problem, a function approximation is used to estimate the values instead of using the value iteration. The deep neural network is used as a function approximation and combined with the Q-learning method. This method is called Deep Q-learning, where the Deep Q-Network approximates the Optimal-Q value [MKS+15]. The DQN model accepts the state as an input and outputs the estimated Q-value for every possible action that can be taken at that given state. Since the optimal-Q value should satisfy the Bellman equation as it is mentioned earlier, Deep -Q learning uses the Bellman equation to find the optimal Q-value by minimizing the loss between the approximated Q-value (output from DQN) and the target (optimal) Q-value (the right-hand side of Bellman equation) for the same action. After calculating the loss, the weights within the neural network are updated by stochastic gradient descent (SGD) just like another neural network. In the Q-learning experience, the replay technique is utilized to train the agent, where the experience of the agent at each time step is stored in the data set called replay memory (D) as a tuple $e_t = <s_t, a_t, r_{t+1}, s_{t+1}>$ where s_t is the current state, a_t is the action that taken at the state s_t, r_{t+1} is the reward that the agent gained at (t+1) by taking the action at the previous (s_t,a_t) pair. Replay memory usually is initialized to have (N) capacity of tuples (experiences). During the learning process, the agent is trained using a random batch of experiences taken from the replay memory. The samples of experience the agent gains and that accrue sequentially in the environment are highly correlated, and that may lead to inefficient learning. The importance of using the "experience replay" technique and training the agent using random samples is to break the correlation between the successive samples of experience. Previously, to calculate the target Q-value, the term $\max_{a'} q_*(s', a')$ in the right-hand side of the Bellman equation which represents the best Q-value in the next time step is obtained from the Q-table. In the case of Deep Q-learning, this term can be calculated by passing the next state (s') to the DQN network as an input which in turn will output the Q-value for every possible action taken from the state (s') and the max value $(\max_{a'} q_*(s', a'))$ can be obtained afterword and plugged to the Bellman equation. Using the same DQN network to calculate the Q-value and the target Q-value will raise a potential training issue since both of them will be calculated based on the same weights, meaning that, every time the Q-value is updated to move closer to the target value, the target value will be updated and moving in the same direction. To overcome this issue a second network is used to calculate the target Q-value and it is called the target network which is a clone of the first DQN. To increase the stability, the weights of the target policy are fixed with the weights of the original network and they are updated to the new weights at every specific number of time steps.

Policy-Based Algorithm

Like the value-based method,the policy-based method selects one possible action and evaluates the agent's behavior thereafter in order to achieve optimization. The essential difference between the two methods is a matter of how to achieve optimality. While the value-based method selects the optimal policy based on the optimal cumulative reward, the policy-based method directly optimizes the policy itself. The policy is parameterizes π_Φ (s,a) and the optimization problem turns out to be finding Φ, which maximizes the policy's objective function J(Φ) [ADBB17]. In other words, policy-based methods learn how these parameters should change the probabilities by which different actions can be taken in different states in order to maximize the expected reward. The main advantage of policy-based methods is their effectiveness for continuous action or the high dimensional space, where the parameters of the 'parameterized policy' are adjusted instead of solving a complicated maximization in every step. The policy gradients (PG) algorithm is widely used to solve the problems of the continuous action space. The policy is represented by a parametric probability distribution (see equation 6.9). In the PG algorithm, the action at state is selected stochastically based on a vector of parameters (Φ), and by adjusting these parameters, the policy is driven in the direction of increasing the cumulative reward [LHP+15]. Policy gradient is the derivatives (vector of derivatives) of the policy's objective function J(Φ) with respect to the parameters (Φ) as shown in equation 6.10 [SMSM99]. The problem can be formalized as shown in equation 6.11, considering (τ) is the agent's trajectory, R(τ) is the corresponding reward, (π_Φ) is the parameterized policy and P$(\tau \mid \Phi)$ is the probability of the trajectory (τ) under the policy (π_Φ). The policy gradients algorithm searches for the local maximum by ascending the gradient of the policy with respects to the parameters (Φ). It seeks to increase the probabilities of the trajectories that give the best return, as shown in equation 6.12. By reformulating the probability of the trajectory P($\tau \mid \Phi$) and decomposing the trajectory into (states – actions), the policy gradients equation can be reformulated as shown in equation 6.13. Instead of integrating over the spaces of both state and action as in the case of stochastic policy gradients, deterministic policy gradients (DPG) integrates only over the state space, which in turns leads to a reduced number of samples, especially in the case of applications with large action states [ADBB17]. DPG is used in the deterministic environment (no uncertainty) where it accepts a state as input and outputs a single action π_Φ (s)=a. On the other hand, the stochastic policy is always needed to explore the complete state-action space. Based on that and for sufficient exploration for the DPG algorithm, the actions are chosen according to stochastic policy behavior, while learning a deterministic target policy. The policy that the agent uses to determine its actions at a given state is called behavior policy, while the policy that the agent uses to update the Q-value is called target policy. Learning the policy can be achieved in two different algorithms, on-policy or off-policy [SLH+14]. In the case of on-policy learning, the behavior policy is the same as the target policy, while they are different in the case of the off-policy learning algorithm.

$$\mathcal{B}\pi_\Phi = P[a \mid s, \mathbf{\Phi}] \tag{6.9}$$

$$\nabla_\Phi J(\Phi) = \begin{bmatrix} \frac{\partial J(\Phi)}{\partial \mathbf{\Phi_1}} \\ . \\ . \\ \frac{\partial J(\Phi)}{\partial \mathbf{\Phi_n}} \end{bmatrix} \tag{6.10}$$

$$\Phi^* = \arg\max_\Phi J(\Phi) = \max_\Phi \Sigma_\tau P(\tau \mid \Phi) R(\tau) \tag{6.11}$$

$$\nabla_\Phi J(\Phi) = E_\tau(\nabla_\Phi \log P(\tau \mid \Phi) R(\tau)) \tag{6.12}$$

$$\nabla_\Phi J(\Phi) = E_\tau(\nabla_\Phi \log \pi_\Phi P(s \mid t)) \tag{6.13}$$

Actor-Critic Algorithm

Actor–critic algorithms combine the benefits of both value-based and policy-based algorithms. The essential idea is that a value function approximator (critic) is used to explicitly estimate the action-value function instead of using the return. These algorithms deal with two different sets of parameters using two different approximators, the critic and the actor. The critic updates the action-value function parameters, while the actor updates the policy parameters based on the direction that is suggested by the critic [JBCG19]. Actor-critic algorithms use an approximate policy gradient as described in equation 6.14, where the $Q_W(s,a)$ is the estimated cation-value function. Deep Deterministic Policy Gradient is a model-free, off-policy, actor-critic reinforcement learning algorithm that searches for the optimal policy that maximizes the cumulative long-term return for the continuous action environment. DDPG uses deep neural network-based approximators [How60]. In the DDPG algorithm, the actor is used to approximate the optimal policy deterministically, which is unlike the stochastic policy, where the policy learns the probability distribution rather than actions. In another words, the output of the actor in DDPG is the best believed action for the given state. After the action is taken by the actor, the critic evaluates that action in order to determine whether the new state is better or worse than the expectation. That can be achieved by maintaining the Q-values of the taken actions towards the target Q-values. Meaning that, the critic learns to evaluate the optimal Q- value based on the output of the actor (best believed action). Like DQN, DDPG algorithm uses the replay memory strategy in order to break the correlation between the data. Generally speaking, DDPG model includes four neural networks-based approximators, actor μ(s), target actor $\mu^{'}$ (s), critic Q(s,a) and target critic $Q^{'}$ (s,a). Figure 6.2 presents an overview of DDPG algorithm. RL has been applied to a variety of autonomous driving tasks, [WCdLF18], [HXXTS17], [CKK+17]. The actor accepts a state (s) as an input, and outputs the corresponding action which maximize the reward, while the target actor has the same structure of the actor. The target actor is updated by the agent every specific number of time step based on the latest values of the actor parameters. The critic accepts state-action pair as input and outputs the expected Q-value, while the target critic has the same structure of the critic. The target critic is updated periodically based on the latest values of the critic parameters. Both target actor and target critic are used to improve the optimization stability

$$\nabla_{\Phi} J(\Phi) = E_{\pi_{\Phi}}(\nabla_{\Phi} \log \pi_{\Phi}(s \mid a) Q_w(s,a)) \tag{6.14}$$

6.2.2 Supervised Learning Compared to Reinforcement Learning

Unlike RL methods where the agent learns by interacting with the environment without any supervision data, in supervised learning, the agent learns using labeled data sets. This means that the expert is explicitly guiding the model on how to act based on the labeled data. In deep neural networks, for example, and during training, the network approximates the future outputs for the observations and then compares them with the labeled ones in order to reduce the error. Supervised learning is mainly dedicated to dealing classification and regression tasks, whereas RL deals with Markov's decision processes, policy learning, and value learning. The simplicity and the speed of the

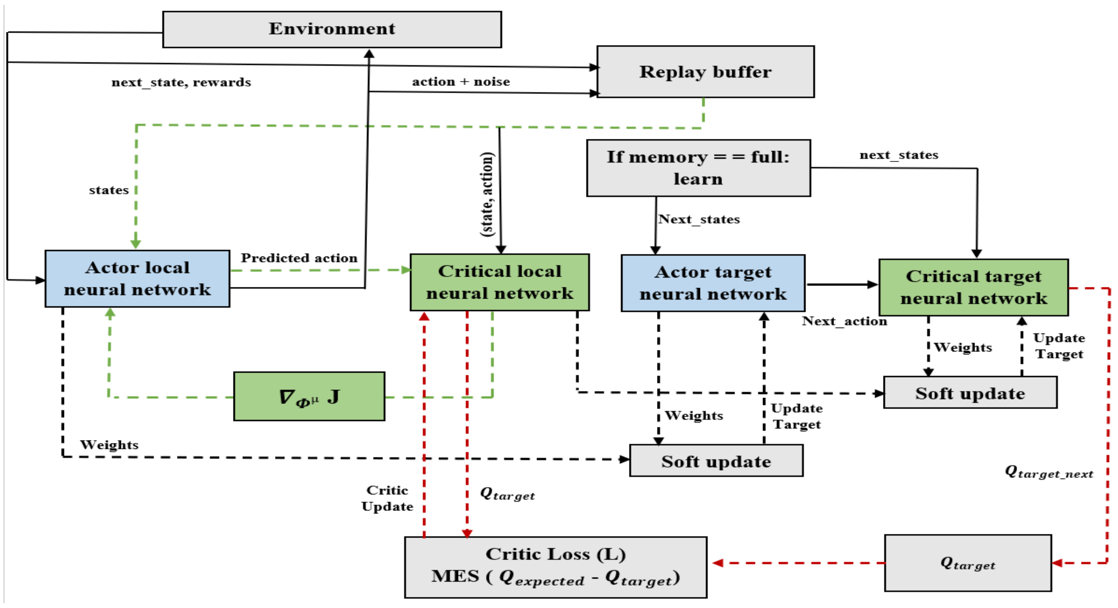

FIGURE 6.2: Overview of DDPG algorithm.

convergence during the training are the advantages of supervised learning compared to reinforcement, where convergence to the optimal policy can be slow so it requires intensive time. On the other hand, the efficiency of the supervised model is greatly affected by the comprehensiveness of the training data-set. In the case of nonlinear and complex systems such as driving system tasks, sufficient training data must be ensured in order to provide an efficient and generalizable model in all complex driving environments. The use of deep learning in a variety of fields has increased recently due to new powerful processing technologies that reduce the training time and improve performance. The deep neural network algorithm is a self-optimization algorithm and it has the ability to adopt a new scenario, which enables the developers to generalize the desired models. These features make deep learning suitable for control applications within dynamic and complex environments. For a better understanding of the importance of successful implementations of the learning-based methods compared to classical ones, it is important to point out some key differences. classical controllers are a model-based control strategies that benefit from the accurate dynamics of the plant, where the model must be explicitly formulated in the control problem. In contrast, learning-based methods have sufficient theoretical assurance, and the efficiency of the model depends on how well the agent is trained. The high computing demands of classical control prompt financial concerns, while offline-trained models require relatively minimal computing time during deployment. Learning-based methods are more efficient in dealing with complex environments with observations obtained, for instance, from photos and videos, which are regarded as more complex for the classical controllers, which deal with signal measurements. Following the standard recommendations during the design process can reduce the computational loads.

6.3 Design of the Controllers

In this section, the designing process of the MPC, the DNN controllers are briefly described where the full details can be overviews in chapter 4 and chapter 5 respectively. On the other hand, the designing process of the reinforcement DQN and the new hybrid models will be descriped in details. DQN and Hybrid models were trained until the determined criteria are achieved (desired reward, number of episodes, ... etc.)

6.3.1 Design of the MPC Controller

Since the MPC is a model-based controller, the first step in the design process is to design the vehicle model. Figure 6.3 shows the global position of the vehicle. Figure 6.4 shows the MPC model, while the input-output signals, the parameters, and the constraints of the MPC are presented in table 6.1. During the designing process, the parameters of the MPC are initiated based on standard recommendations and were tuned during the testing until a stable behavior is achieved. A detailed explanation of the MPC optimization problem, Performance specifications, control law, and parameter calculations can be found in chapter 4.

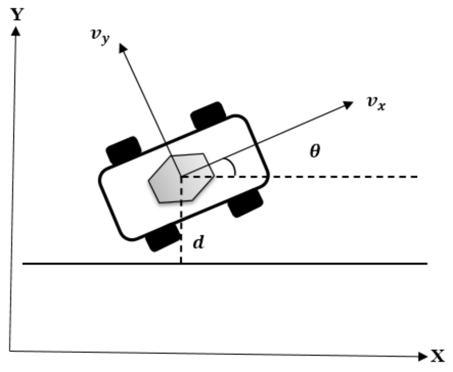

FIGURE 6.3: Global position of the vehicle

TABLE 6.1: Design parameters and system constraints of the MPC controller

Internal model (vehicle)	Input signals	Steering angle (δ)
		Disturbance (ρv_x)
	Output signals	Lateral deviation (d)
		Yaw angle (θ)
		Lateral velocity (v_y)
		Yaw rate (ω)
Parameters of MPC model	Sample time (T_s)	0.1 seconds
	Prediction horizon (P)	2 seconds
	Control horizon (M)	2 seconds
Constraints	Steering angle	[-1.04, 1.04] rad
	Changing rate	[-0.26, 0.26] rad

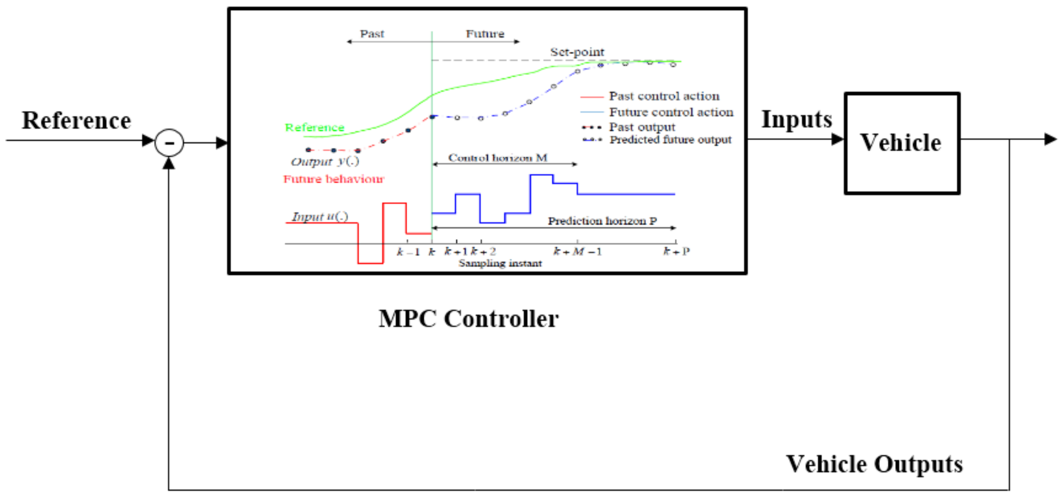

FIGURE 6.4: MPC controller design

6.3.2 Design of the DNN model Using Imitation Learning

To achieve imitation learning, the DNN model was designed and structured based on the MPC model, where six observations are determined as inputs $(\theta, v_y, d, \omega, \rho, \hat{\delta})$ and one control action (δ) was determined as an output, where ($\hat{\delta}$) is the previous control action. The detailed structure is shown in figure 6.5. In regards to the training options, Adaptive Moment Estimation (ADMA) is used as an optimizer, the maximum number of Epoch is set to be 40, the mini-batch for each iteration is set to be 420, and the initial learning rate is set to be 0.01. Data preparation and training process of the MPC controller is presented in chapter 5.

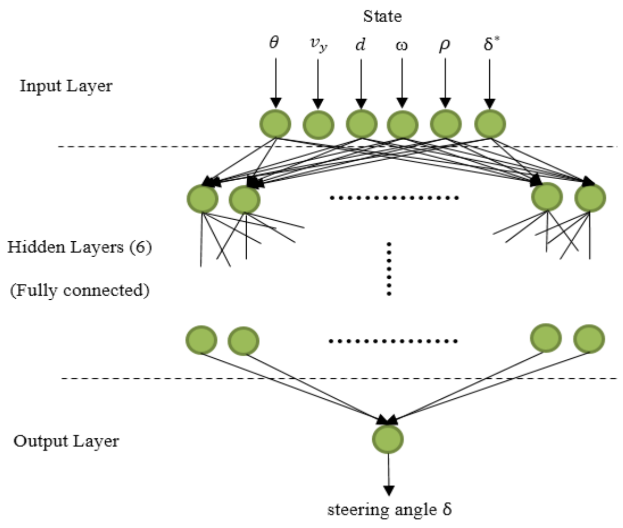

FIGURE 6.5: The DNN model structure

6.3.3 Design of the Reinforcement Deep Q-Learning Model

The desired DQN model is designed taking into consideration the same dynamics of the vehicle, the constraints, and the environment conditions that were used previously. The designing processes went through several steps, preparing the environment, creating and training the agent, and finally testing and evaluating the performance. The environment is created using the six observations and the control action space was determined as a discrete space in the range of [-1.04, 1.04] rad, meaning that the agent can apply 121 possible actions at each state. Based on that, the deep Q-network is designed to accept the state from the environment as an input (vector with 6 observations) and outputs the estimated Q-value of each possible discrete action that can be taken at that state (vector of n=121 Q values). The detailed structure of the DQN model is shown in figure 6.6. The target DQN which is used to calculate the target Q-values is a clone of the DQN with the same structure and parameterization.

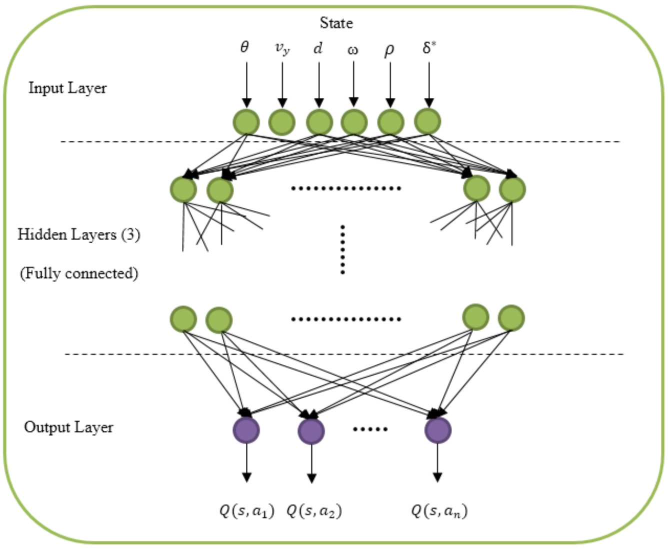

FIGURE 6.6: The reinforcement DQN structure

Training DQN Model

After determining the structure of the DQN model, the training environment is created with corresponding observations, and the training options are determined as follows: ADMA was used as an optimizer, the critical learning rate was set to be 0.001, the total size of the reply memory is set to be 100,000 and the batch size is set to be 64. The sample time and duration for simulation are determined as follows: T_s = 0.1 s and T = 15 s. The training is performed in such a way that each episode lasts at most (T/T_s) time steps. The training is terminated after 40,000 episodes regardless of any other criteria, and it stops when the average of the obtained reward in the current episode is equal to or higher than -1. The agent will be saved during the training if the current reward is equal to or higher than -1.5. By the end of the training, the saved agent is tested and validated by running several simulations in different scenarios within its environment. The training steps can be summarized as follows:

1. Initialize the capacity of the replay memory.
2. Initialize the DQN weights randomly.
3. Create the target DQN network (Clone of the DQN network).
4. For each episode and as long as the termination criteria is not reached:
    (a) Determine the initial state.
    (b) For each time step:
        i. Select and execute an action.
        ii. Obtain the gained reword.
        iii. Observe the next state and store the experience in the memory.
        iv. Sample a batch of agent's experience randomly.
        v. Pass the states to the DQN.
        vi. Pass the next state to the target network.
        vii. Calculate the loss between the outputs (Q-values, target Q-values).
        viii. Update the weights of the DQN to minimize the loss.
        ix. Update the weights of the target network every specific number of time steps.

6.3.4 Design of the Hybrid (Supervised–RL) Model

The same vehicle's dynamics, constraints, environment conditions and state space are used to design and test the new combined model. The continuous actions space is determined to be in the range of [-1.04, 1.04] rad. To create the agent, besides having the trained DNN model as an actor, the critic is created based on the actions-observations specifications, where its neural network is structured to accept seven inputs (state-action) and one outputs (the corresponding expected long-term reward $Q(s, a \mid \Phi^Q)$), and 3 hidden layers. Figure 6.7 shows the detailed structure of the combined model.

Training the Hybrid (Supervised–RL) model

After determining the structure of the suggested mode and creating its environment, the training options were determined (maximum steps, maximum episodes, stop training criteria, etc.) in order to perform the training. The training steps can be summarized as follows:

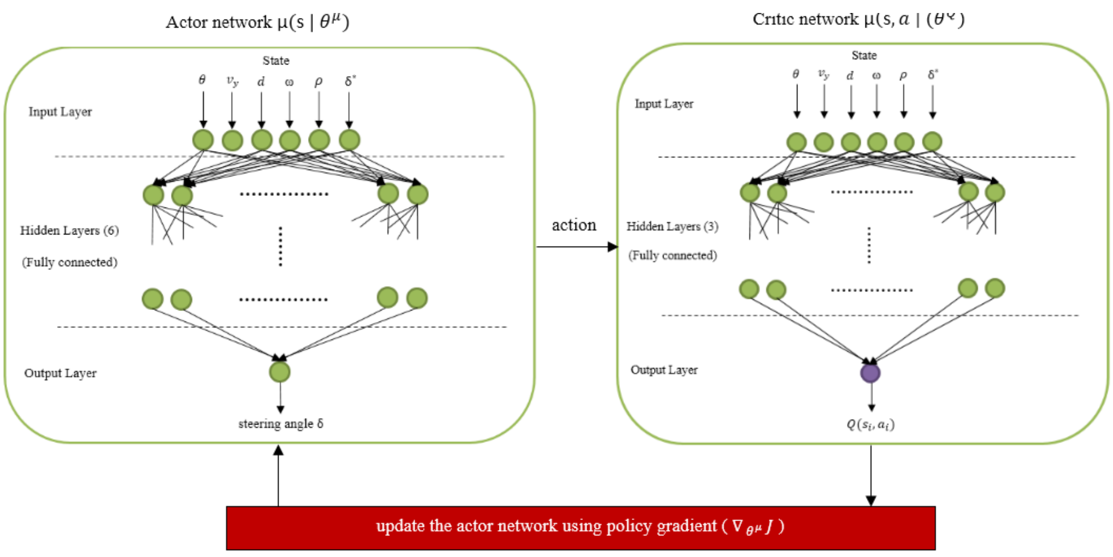
Actor network $\mu(s \mid \theta^{\mu})$
State
θ v_y d ω ρ δ^*
Input Layer
Hidden Layers (6)
(Fully connected)
Output Layer
steering angle δ
action
Critic network $\mu(s, a \mid (\theta^{Q})$
State
θ v_y d ω ρ δ^*
Input Layer
Hidden Layers (3)
(Fully connected)
Output Layer
$Q(s_i, a_i)$
update the actor network using policy gradient ($\nabla_{\theta^{\mu}} J$)

FIGURE 6.7: The structure of the actor-critic networks – combined model

1. Initialize the critic network's weights and target critic network's weights with the same random values.

2. Initialize the actor network's weights and target actor network's weights with the same random values.

3. For each episode and as long as the termination criteria is not reached, do the following at each time step:

    (a) The actor selects an action for the current state.

    (b) Add exploration noise to the selected action to encourage the exploration.

    (c) Observe the reward and the next state.

    (d) Store the experience in the reply memory.

    (e) Sample a batch of agent's experience randomly.

    (f) Pass the next state to the actor and determine the next action.

    (g) Pass the next action to the target critic.

    (h) Calculate the target Q-value (sum of current return and discounted future reward).

    (i) Update the weights of the critic network to minimize the loss between the expected Q-value and the target Q-value.

    (j) Update the weights of the actor network to maximize the expected discounted return.

6.4 Results and Discussion

The implementations of the designed models were performed using the same vehicle model and subjected to the same constraints, environmental conditions, and initial state. The efficiency is discussed based on different indicators: the ability of the controllers to drive the vehicle along the desired trajectory in the first place, the time needed to reach a stable state, and the smoothness of the driving system. In chapter 5, I successfully built a DNN model that imitates the behavior of the classical MPC controller, where the trained DNN and the MPC controller behave similarly with very small output deviation, and the maximum difference is approximately 0.0094 rad (0.53 degrees). Both controllers were able to follow the desired trajectory by driving the lateral deviation and yaw angle to be very close to zero. Additionally, and taking into consideration the control system characteristics, the results clearly show that both controllers were able to reach the stable state at almost the same time with the same amount of overshooting as the figures 6.8, 6.9, and 6.10 show. Based on that, the performances of the suggested hybrid (RL-supervised) model and the reinforcement DQN model are compared to the DNN model in order to evaluate the improvement that is provided by our new hybrid model compared to the other machine-learning-based models (reinforcement DQN and supervised DNN). Figure 6.11 shows that the three models responded differently to the same initial state. Despite these differences, figures 6.12 and 6.13 show that the reinforcement DQN and the combined models were able to track the desired trajectory with different control system characteristics (steady state time and overshooting). The detailed results showed that the combined model responded in a way that improved the smoothness of the driving system by reducing the overshooting (with hardly any overshooting in the case of lateral deviation) and drove the lateral deviation to be very close to zero (0.003 m) in a reasonable time, compared to the DNN model which achieved 0.0009 m as a final value of the lateral deviation at almost the same time but with higher overshooting and thus higher lateral deviations. The DQN model was not as efficient as the other models; its behavior led to higher overshooting and drove the lateral deviation to a final value of 0.01 m. As a result, and taking all the performance indicators into consideration, one can state that the combined model provided the best result and achieved the expected optimization by demonstrating accurate control actions (steering angles) that steer the vehicle along the desired trajectory efficiently in a reasonable time and improve the robustness of the driving system, while the DQN model, which is completely based on an RL algorithm, was not as efficient as the other two models (the supervised DNN or the combined model). The promising results that are provided by the reinforcement learning methods emphasize the importance of devoting more efforts to transferring them into practice as an efficient alternative to classical control methods.

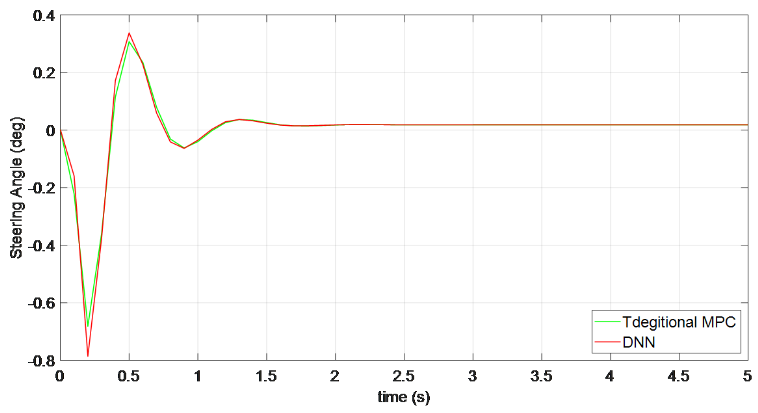

FIGURE 6.8: Comparison of estimated steering angles of MPC and the DNN models

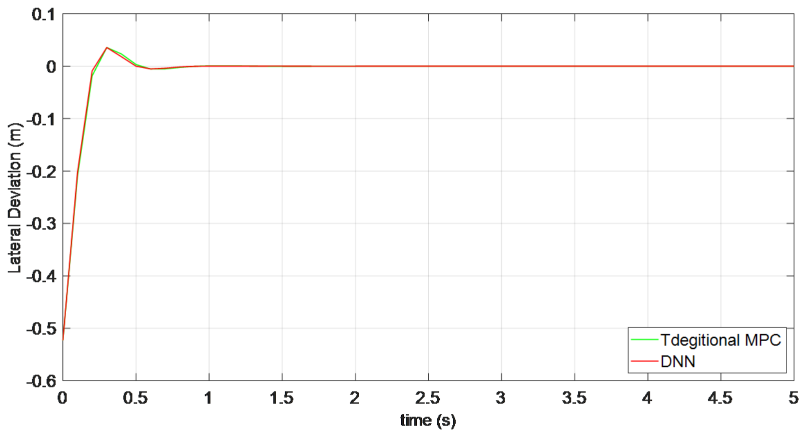

FIGURE 6.9: Vehicle response to the control actions of the MPC and the DNN models – lateral deviation

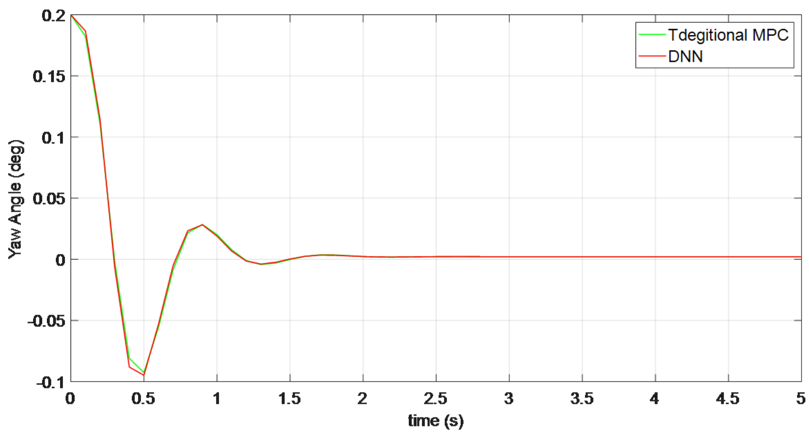

FIGURE 6.10: Vehicle response to the control actions of the MPC and the DNN models – yaw angle

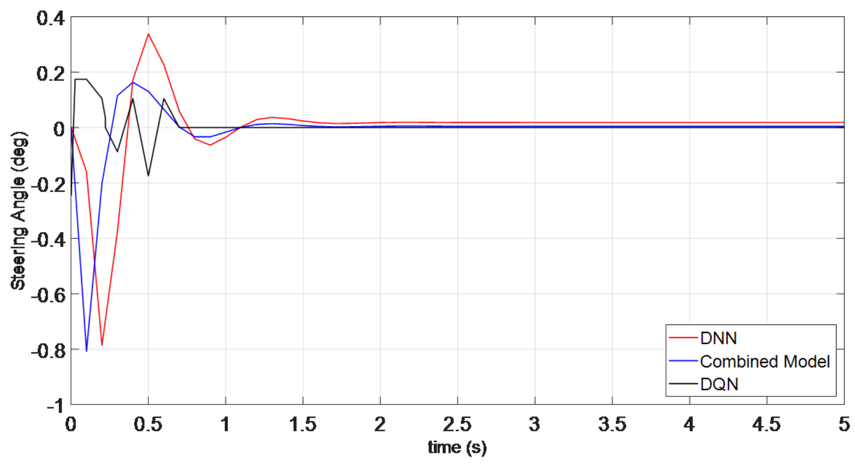

Figure 6.11: Comparison of the estimated steering angles of the DNN, DQN, and combined models

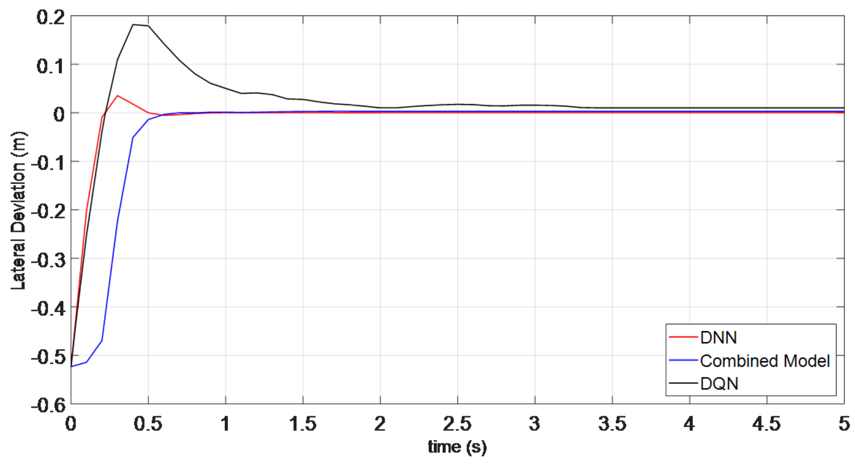

Figure 6.12: Vehicle response to the control actions of the DNN, DQN–lateral deviation

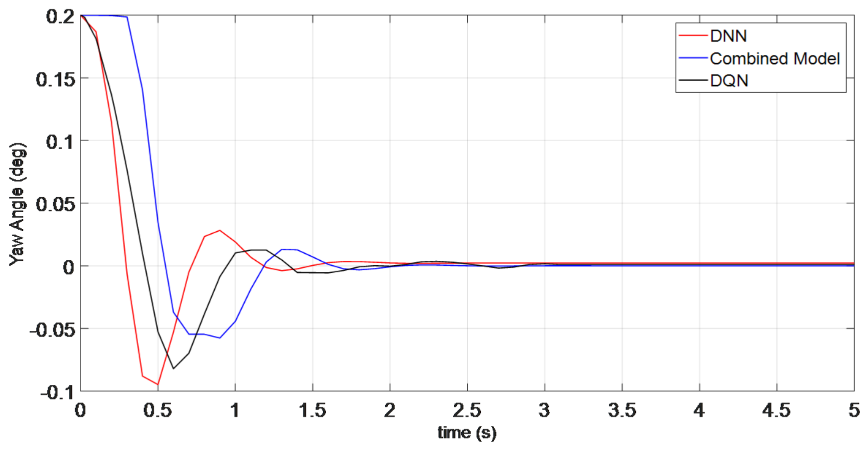

Figure 6.13: Vehicle response to the control actions of the DNN, DQN – yaw angle

Based on the results and discussions, thesis III can be defined:

Thesis III

I proposed machine learning-based control strategy that combines supervised learning (DNN model) and reinforcement learning (RL-model) algorithms in one controller, aiming to achieve the optimization by leveraging the advantages of these algorithms in a way that the RL controller optimizes the actions that are taken by the supervised DNN controller. I evaluated the efficiency of the hybrid model compared to the supervised DNN and reinforcement DQN models which are developed for the same task. The combined model provided the best result and achieved the expected optimization by outputting accurate control actions which reduced the overshooting behaviour, resulting a significant improvement in terms of the smoothness of the driving system.

Related Publications: [ch623], [KRVB20b], [RBV21]

6.5 Chapter Conclusions and Summary

In this work, three different machine learning-based models were designed to perform an automated path-tracking task: a DNN model to imitate the behavior of the traditional MPC controller, a reinforcement learning DQN model, and a hybrid model. The hybrid model was designed to optimize the performance by combining the trained DNN model with the reinforcement learning model, where the DNN network was used as a decision-maker along with the critic network that evaluates the taken actions. The results showed that all three models were able to drive the vehicle along the desired path. The combined model was able to provide the desired optimization by driving the vehicle to the reference target more smoothly and within a reasonable time. This work shows the efficiency of combining supervised and reinforcement learning to leverage the advantages of both algorithms, where supervised learning speeds up the learning process and reinforcement learning improves self-adaptation to new states that the model was not faced with in the training process, which in turns increases efficiency in the complex driving environment.

Chapter 7

Contribution to Automated Driving Safety Using Reinforcement Learning and FPGA Deployments

This chapter concerns the necessity of developing an alternative control approach for safe automated driving that has the capability to deal efficiently with the complexity, non-linearity and uncertainty of vehicle dynamics. Even with the successful implementations of the reinforcement learning in real world for different applications, it is still not commonly used compared to the supervised and unsupervised learning. In this work, a reinforcement learning-based framework is suggested as an alternative solution to the classical control for the application of maintaining a safe distance in autonomous driving system. The efficiency and stability of the suggested model is evaluated compared to the very well known model-based control (MPC) which was developed and implemented for the same task and under the same conditions and constraints. Additionally, in order to verify the efficiency of the suggested model in practice, the trained RL model was deployed and tested on low end FPGA-in-the-loop.

7.1 Introduction

Human perception error comes at the top of the significant factors that cause road accidents. Recently, one of the automotive industry's top priorities has been to provide passengers with the highest level of safety by partially or entirely relieving the driver's responsibilities [MA17]. Over the past decade, control engineers in the automotive industry have devoted major efforts to improving road transportation by developing and applying advanced control strategies. The advancement of sensing, communication, and processing technologies has resulted in a rapid expansion in the development of Advanced Driver Assistance Systems (ADAS) [ZCGC17]. These systems are designed mainly to assist drivers by either offering warnings to lessen risk exposure or automating some control functions to relieve a driver of human control [IWC07]. After being tested in highly structured environments and under full supervision, the autonomous vehicle has been transitioned to the real world alongside human-driven vehicles. It is anticipated that in the near future, fully autonomous vehicles would replace humans in the role of drivers. This progress has raised many critical concerns which need to be addressed. Safety-related challenges are at the top of these concerns, since it is crucial to ensure that the autonomous vehicle is capable of interacting with other vehicles safely. One promising ADAS technology is the Adaptive Cruise Control (ACC), which uses data from sensors such as radar to maintain the speed of the vehicle in order to keep a safe distance from other vehicles on the road. Studies have demonstrated that the safety systems that have been already implemented in the automotive industry such as electronic stability control, ACC and lane following have reduced the traffic accident rate, and consequently increased the safety level [RSB+05].

Generally speaking, any control problem can be described as an optimal control problem and several classical control approaches have been used to address the safety problem of autonomous vehicles. In[NHC+14], Linear Temporal Logic (LTL) is used to describe the system specifications for adaptive cruise control, where a discrete abstraction of the system serves as the foundation for the controller. However, this solution is computationally expensive as the complexity of the controller synthesis is exponential with the dimension of the system (LTL formula). In [AGT14], the safety constraints are maintained as control barrier functions (CBFs), which penalize the violation of the determined constraints, while the control objective (desired speed) is formulated as control Lyapunov functions. These two sets of functions are structured in quadratic problem form. Despite the promising performance, the computational loads and the resource requirement are still very high, and finding the control barrier functions is not an easy task. Model Predictive Control is very well-known control strategy that is used to solve control problems, where it represents the state of art in real time optimal control [AB09]. Due to its capabilities to deal with Multi-Inputs-Multi-Outputs systems and handle multiple (soft and hard) constraints using future prediction strategy. An MPC framework is used to tackle ACC system safety problems, and its efficiency in many other autonomous control applications has already been proven [SdR13]. Bageshwar et al. [BGR04], provided an MPC -based control strategy for transitional maneuvers application. The results show that the efficiency of the MPC controller in achieving a safe and comfortable driving system depends on the explicit incorporation of collision avoidance and acceleration limits into the formulation of the control problem. Using MPC control strategy, multiple control objectives, such as tracking and driver comfort, can be balanced with the cost function by solving the optimal control problem over a finite horizon in a receding fashion [WZZ+18]. However, model-based classical controls such as MPC have a lot of parameters and it takes long time to fine tune as stated in [KAE+12]. In addition, a thorough system dynamics model with high fidelity is required in order to effectively anticipate system states. Solving the optimization problem of high dimensional and nonlinear state spaces causes a significant increase in computational loads, which in turns affects and impedes the real-time implementations, especially with short sample time control. However, linearization is not the best solution to deal with these systems. As a result, the implementations of the classical controllers may become invisible solutions with limited resources computing platforms such as Field Programmable Gate Array and System on Chip. Thus, the necessity of developing an alternative control approach that has the capability to deal efficiently with the non-linearity and uncertainty of vehicle dynamics becomes more urgent.

7.1.1 Background

7.1.2 General Optimization Problem

Optimization is considered an essential aspect in a variety of applications in different areas such as engineering, science and business. The goal of the optimization process differs from one application to another based on the nature of the task itself. Generally, optimization provides improvements in different aspects with the aim of minimizing power consumption and cost or maximizing performance, outputs and profits. The value and necessity of the optimization process mainly come from the complexity of the real-world problems and their limitations in regards of resources, time and computational capabilities. This means that finding solutions to optimally manage and use these resources is an essential topic in the research area. As a result, it is not an overstatement to say that optimization is needed in each and every real-world application. Mathematically speaking, any optimization problem consists of an objective function, decision variables and constraints. The objective function needs to be minimized or maximized and must be subjected to constraints, which can be soft and/or hard constraints. The standard mathematical form of most of the optimization problems can be described as [Ven10].

$$\begin{aligned} minimize f_i(x), &\qquad (i = 1, 2, ...N), \\ g_j(x) \leq 0, &\qquad (j = 01, 2, ...J), \\ g_l(x) = 0, &\qquad (l = 0, 1, 2, ...L) \end{aligned} \tag{7.1}$$

where $f_i(x)$ is the objective function, $g_j(x)$ is inequality constraints, $g_l(x)$ is equality constraints, N is number of objective functions, J is number of inequality constraints and L is number of equality constraints. The objective function can be linear or nonlinear, and similarly, the constraints can be linear, nonlinear or even mixed. The components of the vector $x = (x_1, x_2, \dots x_n)^T$ are called decision variables and can be discrete or continuous variables. The inequality constraints can be written also in the form of (≥ 0) and the cost function can be formulated as a maximization problem [Ven10].

$$\begin{aligned} maximize f_i(x), \qquad & (i = 1, 2, ...N), \\ g_j(x) \geq 0, \qquad & (j = 0, 1, 2, ...J), \\ g_l(x) = 0, \qquad & (l = 0, 1, 2, ...L) \end{aligned} \tag{7.2}$$

The optimization problem can be classified based on the two factors, the number of objective functions and the number of constraints. In terms of the number of objective functions, it can be divided into single objective where N=1 and multi objective where $N \geq 1$. On the other hand, and in terms of number constraints, the optimization problem is divided into unconstrained (j=l=0), inequality constrained (l= 0 and $j \geq 1$) and equality constrained (j= 0 and $l \geq 1$) optimization problems. It is worth pointing out that if both the objective function and constraints are linear then the optimization problem is called a linear programming problem. In the case that both the objective function and constraints are nonlinear, the problem becomes a nonlinear optimization problem. The problem is called quadratic programming (QP) when the objective is a linear–quadratic function and the constraints are affine (linear combinations). In some certain objective functions, we could have a local minimum or global minimum. The point is called the local minimum if the value of the function at this point is equal or smaller than its values at nearby points. When the value of the function at this point is equal or smaller than its values at the feasible points, the point is called the global minimum. Figure 7.1 shows three local minimum points and one global minimum point. Global optimization problems are usually more complex and challenging [Rot17].

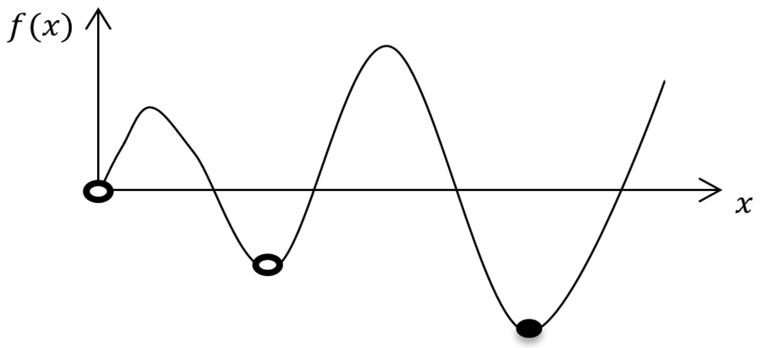

FIGURE 7.1: Local and global minimum points.

7.1.3 Adaptive Cruise Control System

Recently, a cooperative adaptive cruise control (CACC) system was introduced. The main idea of the approach is the inter-vehicle communication, where all the vehicles within the "cooperative team" know the trajectory of each other. However, a collision can happen when an unexpected maneuver occurs due to a communication problem [DYW$^+$15]. Corona et al. [CLSH06], successfully implemented a hybrid MPC framework in order to enhance the safety by achieving an optimal tracking of a time-varying trajectory. The results confirmed the efficiency of the MPC controller in industry with computation restrictions.

7.1.4 Machine Learning Algorithms for Automated Driving

Machine learning algorithms have shown outstanding performance in a wide range of applications in different fields. Machine learning has been used for many autonomous vehicles' applications including perception and motion planning [PSN$^+$17], traffic routing [LYS18], object detection, semantic segmentation [GDDM14a] and others. Pomerleau's autonomous land vehicle was one of the earliest research projects to implement machine learning (neural network) for autonomous vehicle control [Arb60]. Generally, machine learning is divided into three main categories: supervised learning, unsupervised learning, and reinforcement learning. Unlike supervised learning and unsupervised learning, the implementations of the reinforcement learning algorithms are still a very much open challenge [EGCW18]. I presented the RL algorithms in details in chapter 6. An actor-critic reinforcement learning algorithm is suggested in [HXXTS17] for addressing the longitudinal velocity tracking control of autonomous land vehicles. The value function and the optimal policy were approximated based on parameterized features that are learned from collected samples so that the actor and critic share the same linear features. The results show the superiority of the used approach over the classical proportional-integral (PI) control. In [LZLS15], an RL-based model with energy management is suggested to increase fuel consumption efficiency, where the Q-learning algorithm is implemented to obtain the near-optimal policy. In [DCD11], an RL-based controller was suggested to ensure the longitudinal following of a front vehicle, where a combination between gradient-descent algorithm and function approximation is employed to optimize the performance of the control policy. This study concluded that, although performance was good, more steps must be taken to address the control policy's oscillating nature utilizing continuous actions. In [IRC$^+$18], a deep reinforcement learning algorithm is proposed to study the efficiency of RL in handling the problem of navigation within occluded intersections. The solutions discovered by the proposed model highlight various drawbacks of the present rule-based methods and highlighted many areas for more investigation in the future. A multi-objective deep reinforcement learning (DQN) [LC19], is suggested for automotive driving, where the agent is trained to drive on multi-lane ways and in intersections. The results show that the learned policy was able to be transferred to a ring road environment smoothly and without compromising performance. The contribution of this work comes in the orientation of enriching the studies that have been conducted on reinforcement learning method in the frame of safety automated driving in order validate its efficiency and stability compared to the used classical control approaches which in turn, bring it closer to the real world applications. In this work, two different control models were developed for the application of maintaining safety distance between two vehicles, an MPC-based control system and a reinforcement learning-based control system. The performance and the efficiency of the developed RL-based model is evaluated comparing to the behavior of the classical MPC-based model. The two controllers were implemented and tested using the same environment and subjected to the same conditions and constraints. Additionally, the trained RL model is implemented on low end FPGA-in-the-loop in order to verify its performance in practice.

7.2 Design of the Controllers Models

The two controllers are designed to respond to environmental changes using two control modes, speed and maintain modes. In the case where the relative distance (d_{rel}) between the two vehicles (ego vehicle and leading vehicle) is greater than the reference safety distance (d_{saf}), the controller applies the speed mode that makes the ego vehicle drive at the reference velocity. In the case where the relative distance is less than the safety distance, the controller switches to the maintain mode, and the vehicle drives at the speed of the leading vehicle to keep a safe distance (figure 7.2).

7.2.1 Design of the MPC Controller

As the MPC controller is a model-based control strategy and it depends on the feedback coming from its internal plant, the first step in the design process is to design the vehicle model and the driving environment.

7.2.1.1 Vehicle Models and Driving Environment Measurements

The driving environment that is used to test and evaluate the suggested controllers is generated in MATLAB using three main subsystems, the Ego vehicle, the lead vehicle and

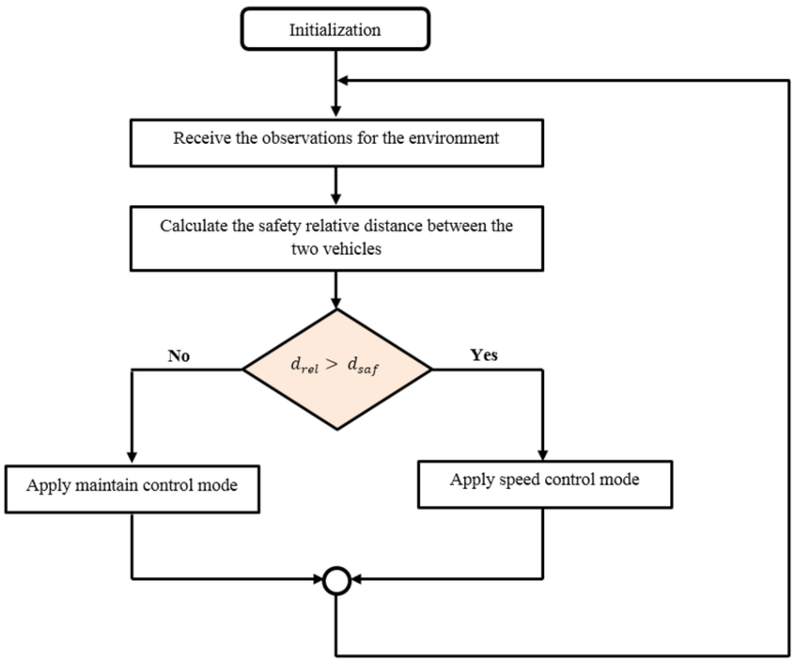
Initialization
Receive the observations for the environment
Calculate the safety relative distance between the two vehicles
$d_{rel} > d_{saf}$
No
Yes
Apply maintain control mode
Apply speed control mode

FIGURE 7.2: The schema of applying the control modes.

controller modules. Based on the relative velocity and relative distance between the two vehicles, the controller makes decisions to increase or decrease the speed of the ego vehicle in order to maintain the safety distance. The relative velocity and relative distance are obtained by taking the difference in current velocity and the difference in current distance between the two vehicles respectively as figure Figure 7.5 shows. The relation between the acceleration and the longitudinal velocity is subjected to the dynamics that describe the relation between the engine throttle system and the resistance of the vehicle to the change in direction or velocity (vehicle inertia). The throttle system regulates how much air and fuel mixture reaches the engine cylinders using a control valve. By increasing the opening angle of the control valve, more fuel mixture enters the engine, which in turn increases the vehicle speed. Equation 7.3 describes this relation, where TF is the transfer system that approximates the dynamics between the throttle system and the vehicle inertia and S is the complex variable in Laplace transform. Figure 7.3 represents the mathematical relation between acceleration, velocity, and position of the vehicle, which clearly shows that, changing the acceleration leads to change the velocity and the position of the vehicle. Based on that,the initial state of the two vehicles were determined at the beginning of the test process, including the initial positions, the initial velocities and the initial accelerations. During the test, the acceleration of the lead vehicle was changed continuously using a dedicated generator (MATLAB function) in order to change the state of the lead vehicle (velocity, position), which in turn changed the relative distance and the relative velocity between the two vehicles, so the controller had to response to many different scenarios during the test.

$$TF = \frac{1}{S(0.5S + 1))} \tag{7.3}$$

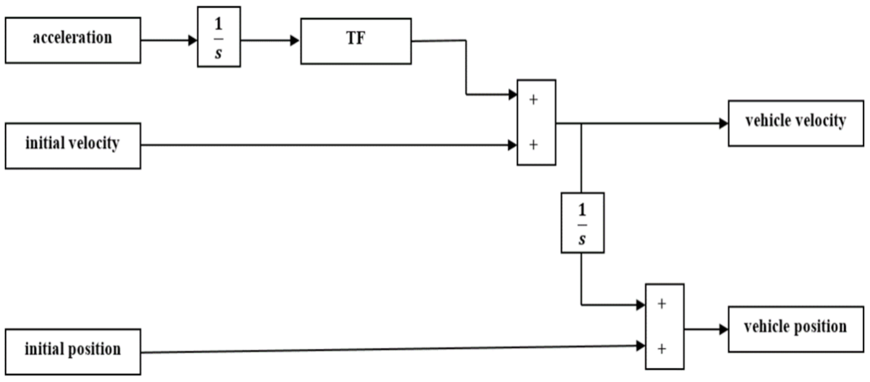

FIGURE 7.3: MATLAB simulink subsystem -vehicle model actual position and velocity.

7.2.1.2 Design Parameters of MPC Controller

After designing the plant model, the next step is to determine the input/output signals and the design parameters of the control system (figure 7.4 and table 7.2). The measured outputs are used for state estimation, while the manipulated variables are the optimal control actions. The MPC design parameters and the control constraint are presented in table 7.1. Figure 7.5 shows the overall workflow of the MPC-based control system.

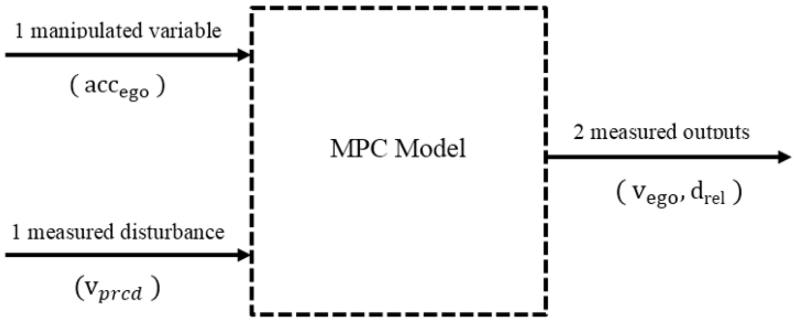

FIGURE 7.4: MPC plant model- Inputs/outputs.

TABLE 7.1: MPC design parameters.

MPC controller parameters	
Sample time (T_s)	0.1 s
Prediction horizon (P)	30
Control horizon (M)	3
Control action constraints	
Acceleration	[-3, 3] m/s^2

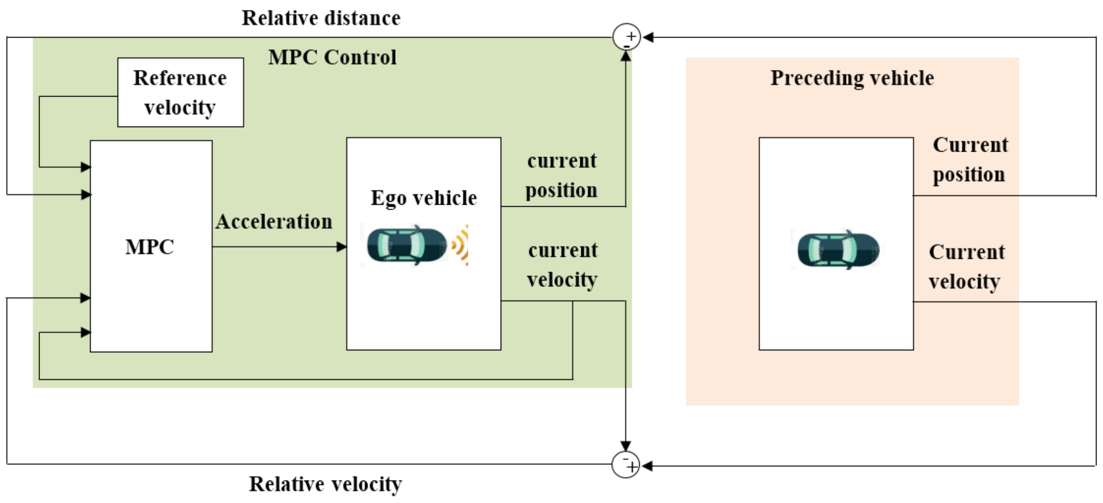

FIGURE 7.5: Overall diagram of the control system-MPC model.

TABLE 7.2: Inputs/Outputs signals of MPC controller.

Signal type	Parameter	unit	Description
Measured outputs (MO)	v_{ego}	m/s	Longitudinal velocity of the ego vehicle
	d_{ref}	m	The relative distance between the proceeding and the ego vehicles
Measured disturbance (MD)	v_{prcd}	m/s	Longitudinal velocity of the proceeding vehicle
Manipulated variable (MV)	acc_{ego}	m/s^2	acceleration\deceleration
References	v_{re}	m/s	Reference velocity in speed mode
	d_{saf}	m	The reference safety distance

7.2.2 Designing and training the RL-Based Controller

In this work, the Deep Deterministic Policy Gradient algorithm is used to design the reinforcement leaning controller. As mentioned earlier, this algorithm uses two different deep learning-based approximators, the actor and the critic. The design process went through several steps, starting from preparing the environment, designing the neural networks of the actor and critic, creating and training the agent, and finally running, testing and validating the model. The observations of the environment are determined to be the vehicle velocity (v_{ego}), the velocity error (v_{er}) and the integral velocity error (p_{er}). Velocity error represents the difference between the reference and the vehicle velocity. The acceleration constraint is determined to be in the same range as that of the MPC controller, [-3, 3] m/s^2. Figure 7.6 shows the neural network structure of the actor-critic approximators. In detail, The Critic network consists of 2 input layers, 4 hidden layers and 1 output layer. The first input layer is structured with three neurons to receive the three observations from the environment, while the second input layer is structured with 1 node to receive the action from actor network. Each hidden layer consists of 50 neurons and ReLU is used as an activation function between the layers. The output layer consists of one neuron to output the estimated Q-value. The actor network consists of 1 input layer with three neurons to receive the observations, 5 hidden layers, each one is structured with 50 neurons, and ReLU activation function between the layers. Figure 7.7 shows the overall workflow of the RL-based control system.

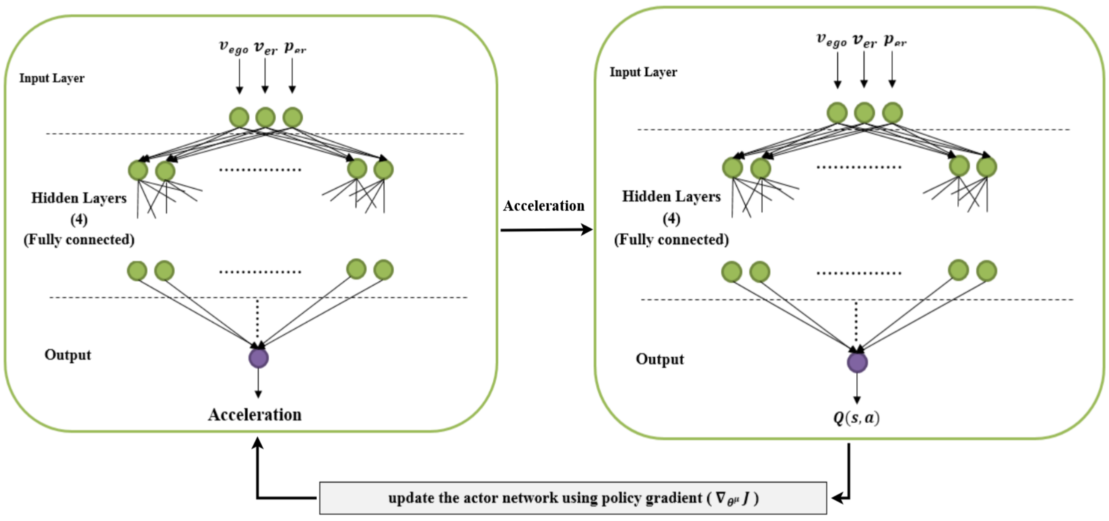

FIGURE 7.6: RL model: actor – critic neural network structure.

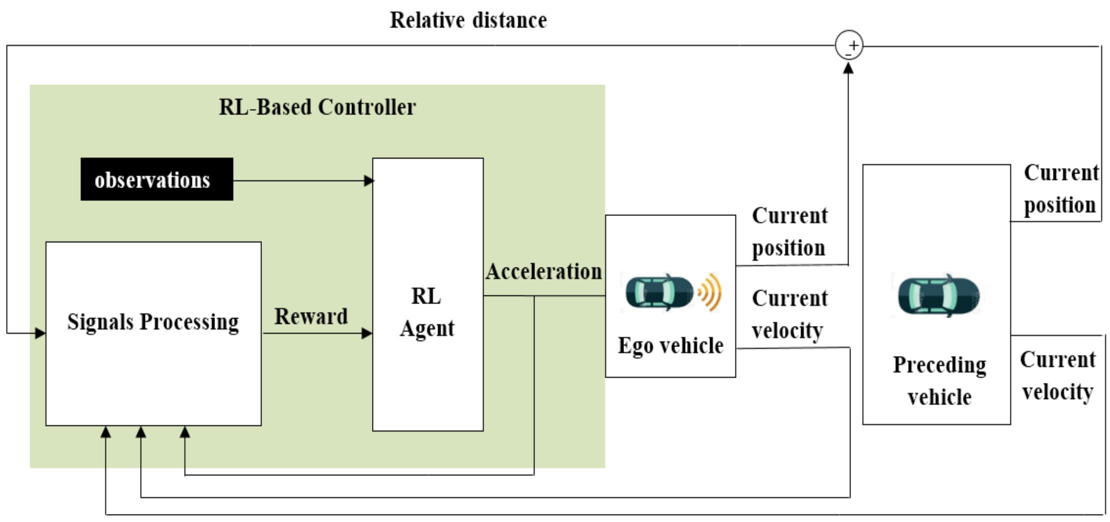

FIGURE 7.7: Overall diagram of the control system-RL model.

Training Algorithm

The goal of the training is finding the optimal policy which achieves the highest cumulative rewards. During the training, the parameters of the policy is updated based on the following algorithm: taking action, getting the reward, and update the policy. This cycle continues until the trained agent learns to take the best action that achieves the highest long-term reward at each state. The training options were determined such as the maximum number of episodes, the number of steps per episode and the stop training criteria. Considering S, A represent the state and the action respectively, training algorithms can be summaries as the figure 7.8 shows.

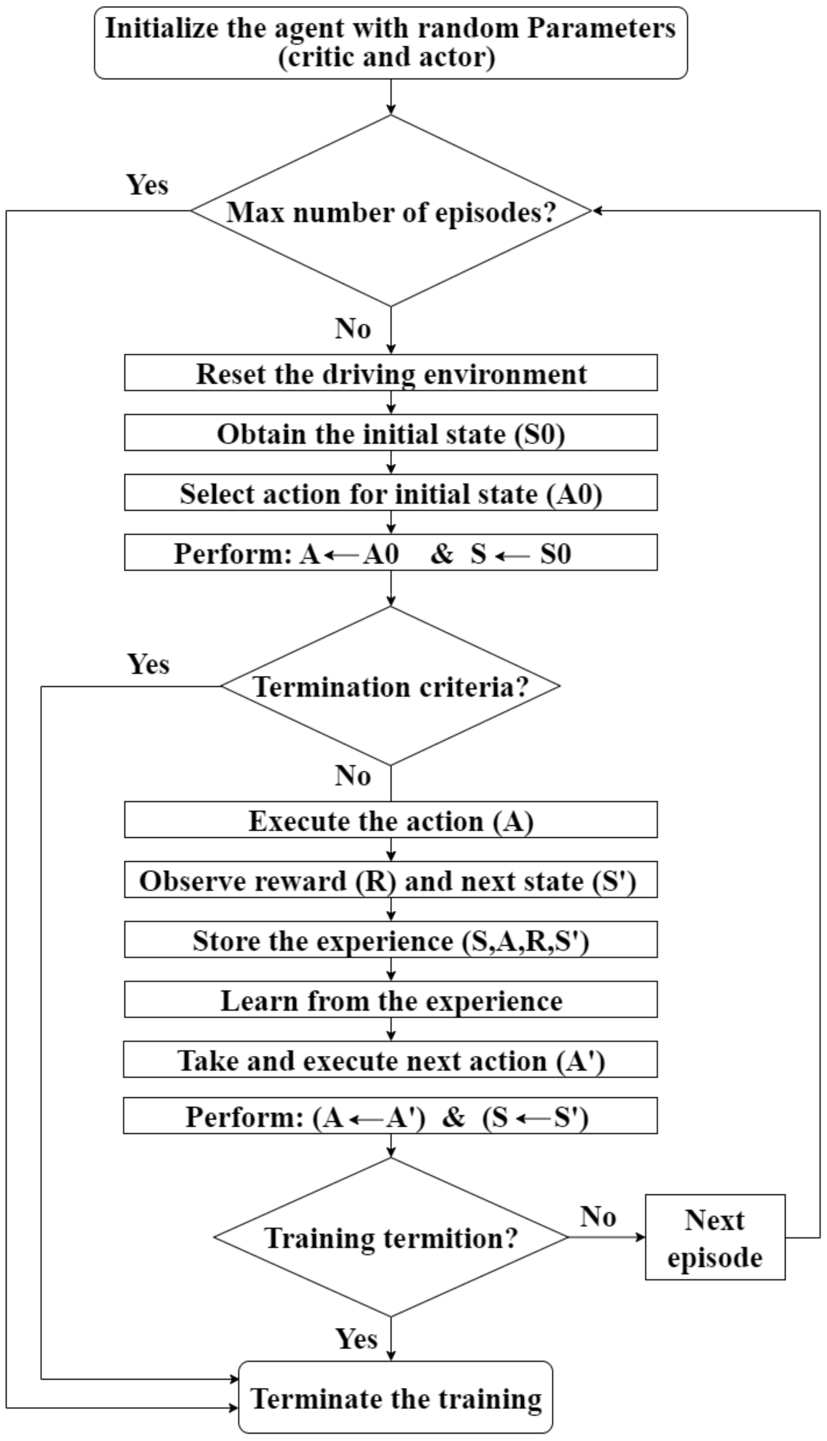
Initialize the agent with random Parameters (critic and actor)
Max number of episodes?
Yes
No
Reset the driving environment
Obtain the initial state (S0)
Select action for initial state (A0)
Perform: A ← A0 & S ← S0
Termination criteria?
Yes
No
Execute the action (A)
Observe reward (R) and next state (S')
Store the experience (S,A,R,S')
Learn from the experience
Take and execute next action (A')
Perform: (A ← A') & (S ← S')
Training termition?
No
Next episode
Yes
Terminate the training

FIGURE 7.8: The overall diagram of running RL model on FPGA.

7.2.3 SOC Implementations and Verification of RL model

Due to the fact that the reinforcement learning algorithm depends on trial and error, it is safer to test the design using the MATLAB Simulink first, before the implementation on the real SOC. For this purpose, multiple simulations were performed and the final trained policy that meets the requirements is taken to the next step to be deployed on the target FPGA. The suggested RL-based control model is deployed on a low-end SOC (ZedBoard). The deployment process went through several steps (see figure 7.9). In the first step, the trained policy is extracted from the trained agent (the Simulink model) and represented in C code generated using MATLAB Embedded Coder. The generated code accepts the environment observations as inputs, and outputs the optimal action for the current state based on the trained policy. In the next step, the hardware configurations were prepared in order to perform the communication between SOC and MATLAB Simulink. The deployment is performed by downloading and running the generated code on the target SOC. Running the model on Zedboard goes through the following cycle: SOC receives the signals from MATLAB Simulink through the communication channel, executes the algorithm, outputs a control action (acceleration or deceleration), and then sends it to the Simulink model to update the state of the driving environment.

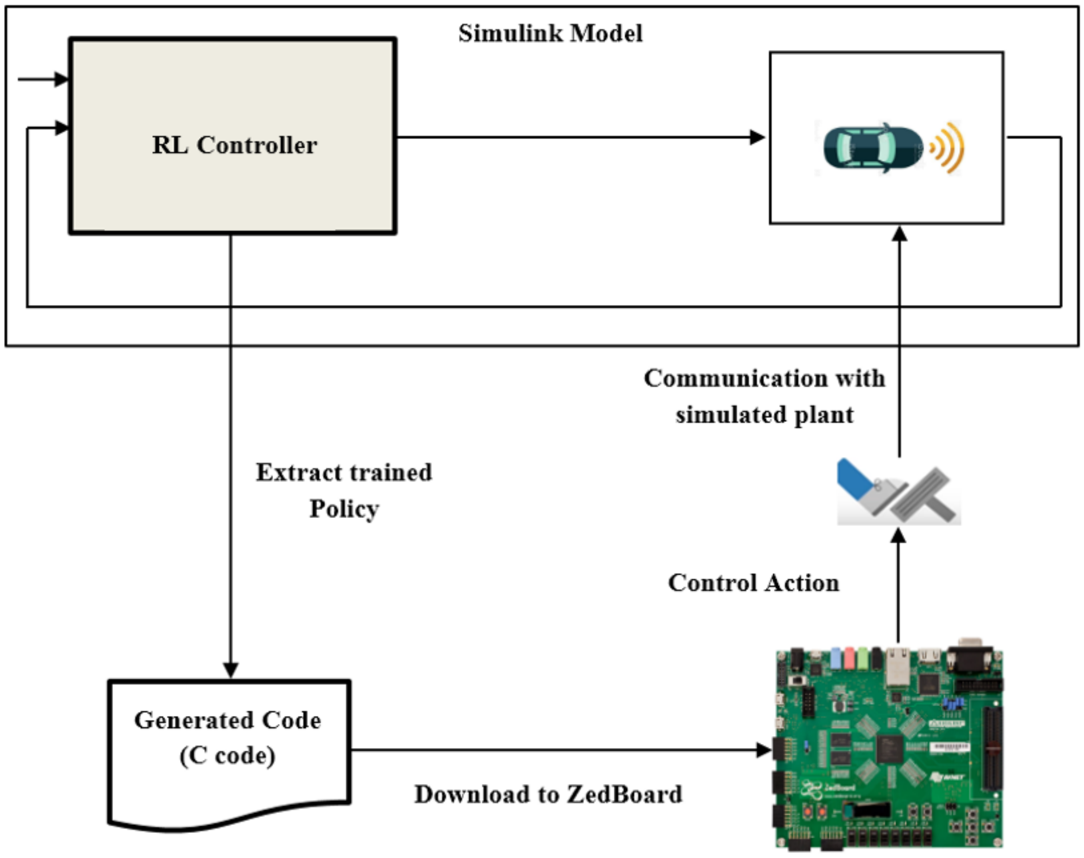

FIGURE 7.9: RL training algorithm.

7.3 Results and Discussion

In the first part of this section, the performance of the RL model is evaluated and compared to the performance of the MPC model. In the second section, the deployment of the RL-based model on SOC is discussed and evaluated compared to the Simulink results. The efficiency of the models is determined based on different metrics: the ability to follow the environment changes, the response time, and the overshooting, and the ability of the controller to switch between the two control modes (speed and maintain modes) in a way that the vehicle either keep the safety distance by following the preceding vehicle or follow the reference speed based on the current state of the environment.

7.3.1 MATLAB Simulink

Figures 7.10 and 7.13 show the behavior of the MPC and RL-based controllers, respectively, and their response to different driving scenarios. Figures 7.11 and 7.14 show the relative distance between the two vehicles compared to the reference safety distance. Figures 7.12 and 7.15 show the change of the preceding vehicle's velocity, the reference velocity and the response of the ego vehicle to the outputs of the controllers.

7.3.1.1 MPC Controller

Figure 7.11 shows that, in the first 10 seconds, the relative distance between the two vehicles is greater than the safety distance, and the MPC controller response is based on the speed control mode, where it successfully drives the ego vehicle to follow the reference velocity (figure 7.12). Between 12 to 28 seconds, the relative distance is less than the safety distance, the MPC controller switches to "maintain" control mode, and the ego vehicle changes its speed in order to follow the minimum speed of the preceding vehicle. The detailed behavior of the MPC during this period of time (0 to 28 seconds), shows that in the first three seconds, the MPC controller outputs a near maximum acceleration (figure 7.10) since the relative distance is much higher than the safety distance and the ego vehicle followed the reference speed. On the other hand, with the continuous decrease in the velocity of the preceding vehicle and consequently the relative distance, MPC begins to gradually reduce the acceleration. At the second 10 the relative distance became very close to the safety distance, and the MPC controller switches to the "maintain" control mode, and responds in a way that drives the vehicle in the direction of following the speed of the preceding vehicle. At second 28 the MPC controller switches back to "speed" mode and the ego vehicle follows the reference speed until second 58 regardless of the (maximum or minimum) speed of the preceding vehicle, since the relative distance is greater than the safety distance.

7.3.1.2 Reinforcement Learning Control Model

Figures 7.13, 7.14, and 7.15 show that the RL-based controller was able to switch between “speed” and “maintain” control modes efficiently. The controller drives the ego vehicle to follow the reference speed between 0 to 10 seconds, since the relative distance is greater than the safety distance. Then it successfully switches to “maintain” control mode when the relative distance becomes very close or less than the safety distance between 10 to 28 seconds. From 31 to 56 seconds the relative distance is greater than the safety, and the RL controller switches back to “speed” control mode, where the ego vehicle follows the reference speed.

7.3.1.3 Performance Comparison

The results show that both controllers responded efficiently to the environment’s changes based on the control modes and the design specifications. In detail, at the beginning the RL controller drove the vehicle from the initial state to the reference velocity in approximately 4.5 s, while it took around 6.2 s in the case of MPC controller. The results also show that the RL controller was able to follow and respond to the environment changes faster than the MPC controller: it took approximately 4.6 s to reach the preceding vehicle speed after changing to “maintain” mode with the RL controller, while it took approximately 6.4 seconds in case of MPC controller. Similarly, the results showed a stable behaviour of the RL model which responded faster to the other environment changes, where it drove the vehicle at the reference speed at second 26 s after switching back to the speed control mode, while the MPC controller reached the reference speed at second 28.5 s. RL-based model improved the response time with 1.75 s in average. On the other hand, the behavior of the RL controller shows higher overshooting in terms of following the reference speed compared to the MPC behavior, especially against the initial state, where the maximum overshooting was approximately 1.3 m/s. As a conclusion, the results demonstrate the advantage of the reinforcement learning model in term of the ability to predict and follow the changes in the environment state faster than the MPC controller. This result is compatible with the fact that solving the optimization problem using a trained neural network is faster than solving an online quadratic problem at each time step (MPC controller). On the other hand, the results demonstrate the advantage of the MPC control in term of providing less overshooting behaviour in all switching points.

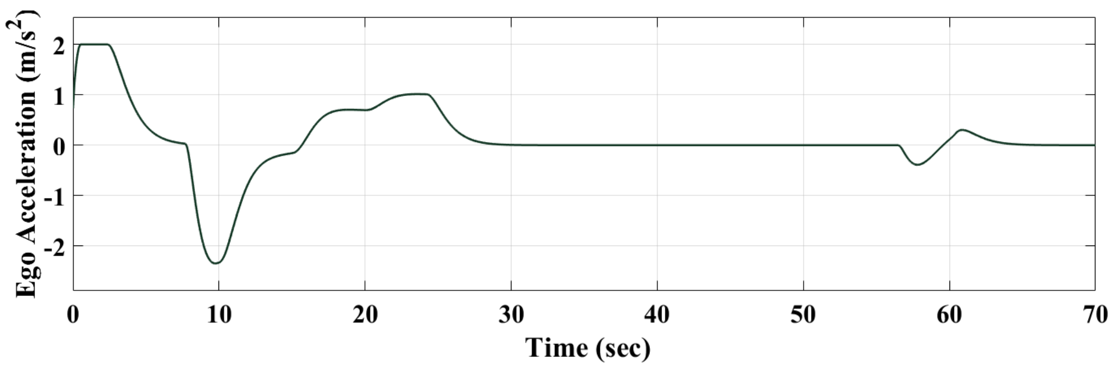

FIGURE 7.10: MPC-based controller's response – MATLAB Simulink.

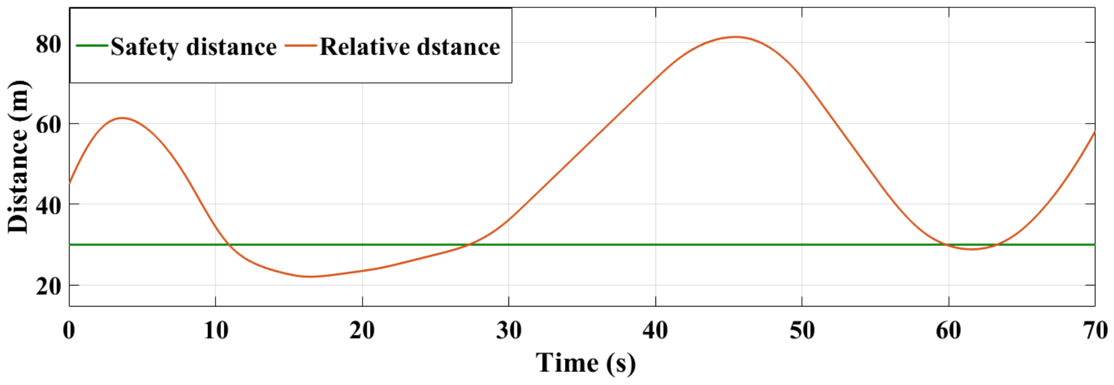

FIGURE 7.11: Relative distance compared to safety distance – MPC controller.

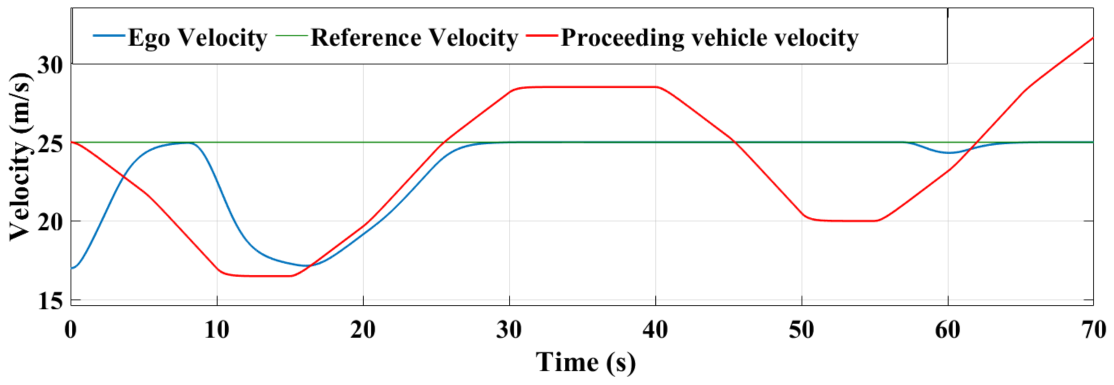

FIGURE 7.12: Velocity of ego and proceeding vehicle compared to reference velocity – MPC controller.

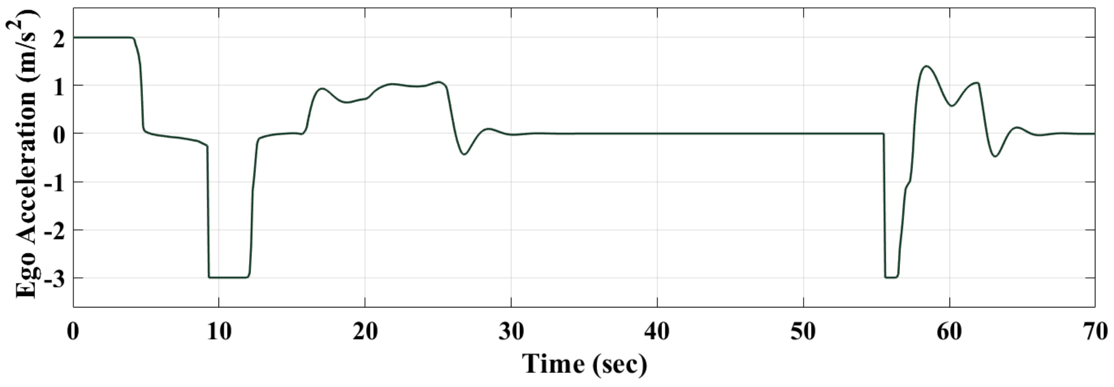

FIGURE 7.13: RL-based controller's response – MATLAB Simulink.

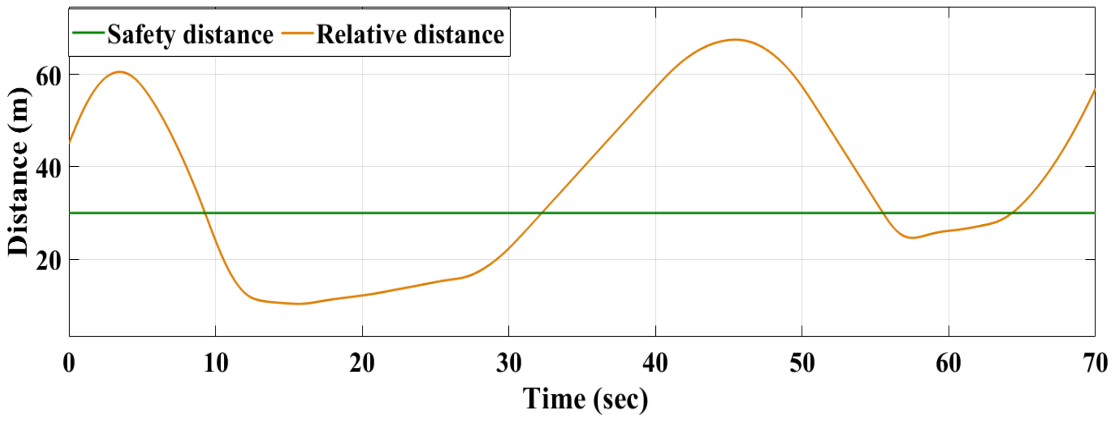

FIGURE 7.14: Relative distance compared to safety distance – RL controller.

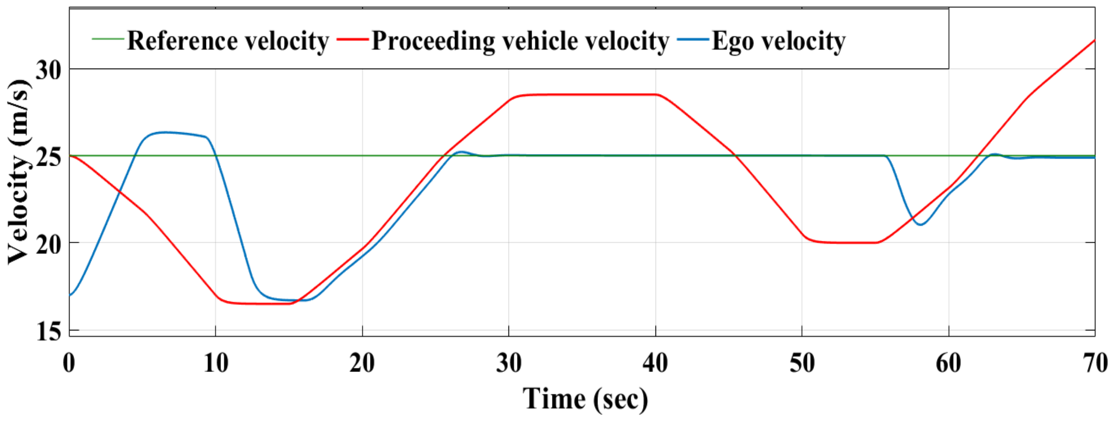

FIGURE 7.15: Velocity of ego and proceeding vehicle compared to reference velocity – RL controller.

7.3.2 RL-model on SOC

The performance of the RL controller after being implemented on SOC is evaluated compared to its performance on MATLAB Simulink. The compression is analyzed in term of tolerance error, which is determined to be 0.2 m^2/s maximum. The results in figure 7.16 shows that the response of the RL controller is within the determined tolerance range, and figure 7.17 represents the detailed deviations where the maximum was $(4.9e^{-06}m^2/s)$. These results demonstrate the success of the deployment of the proposed RL controller on SOC, which also in turn indicates the effectiveness of the approach employed to extract the optimal policy that is trained and validated using MATLAB Simulink.

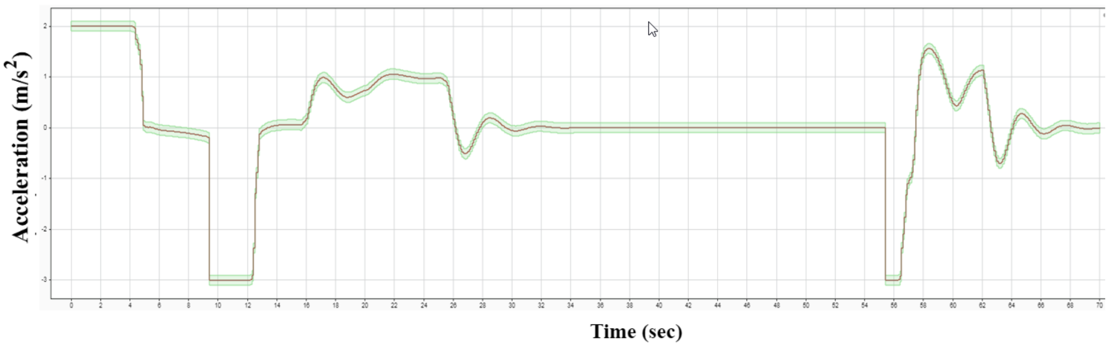

FIGURE 7.16: FPGA – Simulink comparison - RL-based controllers' response within the tolerance range.

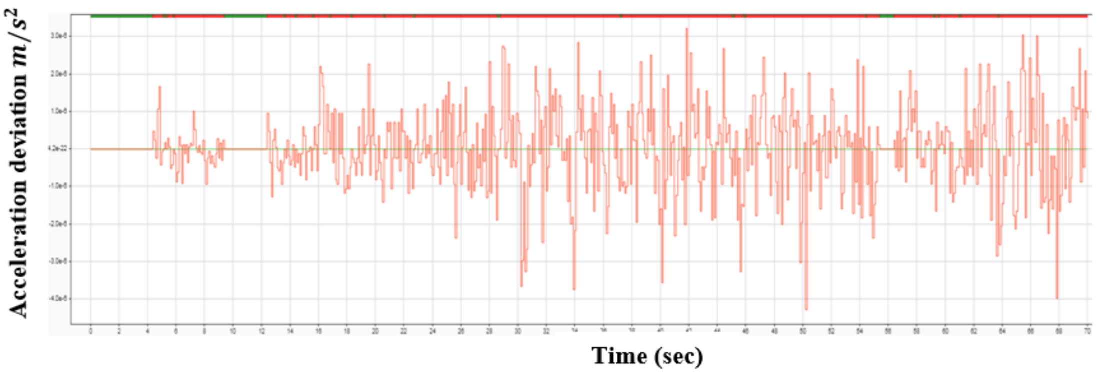

FIGURE 7.17: FPGA – Simulink comparison – RL-based controller's response deviations.

Thesis IV

I proposed a reinforcement learning-based control strategy for the task of maintaining a safe distance in the frame of an automated driving system. I also proposed a method to deploy the developed model on low-end FPGA. The method extracts the policy of the trained RL agent and converts it to a C code that is downloaded and run by the target SoC (FPGA). Compared to the traditional MPC controller, the results show the superiority of the reinforcement learning-based model in terms of the ability to predict and follow the changes in the environment states and improvement in response time (1,75 s in average). This work contributed to enriching the research on RL algorithms and paving the way to bring it closer to practice.

Related Publications: [RV23]

7.4 Chapter Conclusions and Summary

In contrast to the classical control strategies, whose efficiency strictly depends on the design parameters, the reinforcement learning algorithm provides an efficient generalizable model and higher stable control using a very well-trained agent that is exposed to all possible scenarios within the environment. In this work, a reinforcement learning-based control strategy was suggested for the task of maintaining a safe distance in an automated driving system as an alternative solution to the classical model-based control. The performance of the suggested model is evaluated compared to the performance of MPC controller that is designed for the same task. Additionally, the RL model is deployed on a low-end FPGA/SOC after being verified in MATLAB Simulink. The obtained results show the superiority of the reinforcement learning algorithm over the MPC in terms of the ability to predict, follow and respond faster to environment changes. RL model provided a stable behaviour against the environment changes, where it improved the response time with 1,75 s in average. On the other hand, the MPC controller outperformed the RL-based controller performing with less overshooting (approximately 1.3 m/s less) which demonstrate the advantage of the MPC control in term of overshooting behavior. After verifying the reinforcement learning model, the trained policy is extracted and successfully deployed on FPGA.The obtained results showed a stable behaviour of RL model after being implemented on FPGA compared to the Simulink behaviour, where its response was within the tolerance range (0.2 m^2/s) and the maximum deviation was ($4.9e^{-06}m^2/s$). In conclusion, developing a reinforcement learning-based control strategy for highly complex environments can be considered as a promising and efficient solution to address the disadvantages of classical control approaches.

Chapter 8

Summary

8.1 Contributions

Given the great importance of the autonomous vehicles in reducing the traffic risks and accidents on the roads resulting from human errors, engineers in the automobile sector have worked extremely hard over the past decade to develop and implement control strategies in an effort to improve road transportation and reach a fully autonomous and safe vehicles. Without a doubt, the next few years will witness a significant increase in fully autonomous vehicles on the roads. The main motivation of my doctoral work is to contribute to this efforts by making use of the advanced technologies and algorithms to achieve the desired optimization. This contributions can be summarised into four main parts. It is known that the efficiency of the classical control strategies drops off in highly complex environments due to their weakness in handling the dynamically changing systems and high processing demands, especially when it comes to the limited resources of embedded computing platforms such as system-on-chip and field-programmable gate array. In the first part, I addressed these problems for model predictive control as on of the most traditional control strategies that is used for automated steering task. The aim was to optimize the implementations of MPC controller. To deal with dynamic changes systems, the adaptivity concept was used, where the optimization problem remains the same (same number of states and constraints) but a new linear model is used at each time step to obtain an accurate prediction based on the new conditions, and that is called adaptive model predictive control. The obtained results showed that adaptive MPC was able handle the chaining dynamics and yields a good performance in terms of tracking the reference trajectory. In the second section of this work, I applied functional on-target rapid prototyping using embedded coder and HDL coder (hardware/software co-design) for the implementation of embedded systems dedicated for digital signal processing considering performance, execution time and resources consumption. The suggested implementation method is based on taking the optimization problem of the control method through MATLAB Simulink, Fixed-Point Designer, Embedded Coder and HDL coder, which allows the authors to focus on the verification, the validation and the test of the embedded system rather than programming, which in turn gives the ability to refine the design. Different strategies were implemented to achieve the resource optimization, where the implementations involve Logical optimization, placement of logic cells, and routing the connections between cells. In the second part, I worked on developing DNN-based controller as alternative to the classical MPC in the frame of automated driving task, suggesting that the use of a deep neural network can significantly increase efficiency and inevitably result in reduce the time and the complexity of implementations. The suggested DNN model, was designed and trained to imitate the behaviour of the traditional MPC. Considering the crucial function that deep neural network hardware implementations play, a new automatic IP generator has been developed in order to deploy and optimize the implementations of the DNN based-models on Field Programmable Gate Array. The performance of the suggested DNN controller after being deployed on low end FPGA was evaluated, and the obtained results show that it is successfully imitated the behaviour of the classical MPC, provided a good performance, and improved the execution time (up to 4 times faster). In the third part, I suggested a new hybrid machine-learning model that combines two controllers (DNN-based controller and RL-based controller) in one model. The idea behind the hybrid model is to leverage the advantages of the supervised learning and reinforcement learning algorithms in one control model in a way that the RL controller optimizes the actions that are taken by the supervised DNN controller that is developed in chapter 5. The efficiency of the hybrid model was evaluated compared to the supervised DNN and reinforcement DQN models. The obtained results show that the combined model was able to provide the desired optimization by driving the vehicle to the reference speed more smoothly and within a reasonable time. These results proved that the suggested hybrid mode has better generalization capability within the complex driving environment. The supervised learning speeds up the learning process while the reinforcement learning improves self-adaptation to new states that the model was not faced with in the training process. Although the promising results achieved by the implementations of reinforcement learning algorithms in different field, RL still an emergent field and the real-world applications still very much open challenge and has not yet been applied to practice as successfully as supervised and unsupervised learning. In this context, in the fourth part of this work, I suggested a reinforcement learning-based framework as an alternative solution to the classical control for the application of maintaining a safe distance in autonomous driving system, which enriching the research on RL algorithms and paving the way to bring RL closer to real-world. Additionally, RL model was deployed on a low-end FPGA/SOC after being verified in MATLAB Simulink. In order to perform the deployment, first I extracted the policy from the agent after being trained and I represent it as a C code which was downloaded and tested on SOC. The obtained results showed that the RL model was able to handle the environment changes more efficient compared to MPC controller and improved the response time with 1.75 s in average. Additionally, the method that I used to perform the deployment is evaluated, and the results showed the efficiency of this method, where the RL model provided a stable behaviour after being implemented on FPGA compared to the Simulink behaviour. Table 8.1 summaries behaviour comparison between the suggested machine learning-based controllers and the traditional MPC in the frame of automated driving task, taking into concentrations the overshooting, settling time, and the execution time as reference indicators. All the controllers were evaluated under the same environment, initial state, conditions, and constraints.

TABLE 8.1: Summary of the suggested ML-Based controllers compared to traditional MPC for the task of automated driving

Application	Controller	Observation	Overshooting (m)	Settling Time (s)	Execution Time (s)
Automated driving	MPC Controller	Lateral deviation	0.036	0.60	1.54
		Yaw Angle	0.093	1.05	
	DNN-Based Controller	Lateral deviation	0.035	0.61	0.34
		Yaw Angle	0.094	1.03	
	DQN - Based Controller	Lateral deviation	0.182	1.3	0.37
		Yaw Angle	0.082	1.2	
	Combined Controller (DNN-RL)	Lateral deviation	0.0034	0.5	0.23
		Yaw Angle	0.0576	1.3	

8.2 Theses

Thesis I

I gave a methodology for adaptive MPC development for the changing dynamics system, which significantly improved the behaviour of the controller in terms its ability to predict and follow the dynamic changes of the system efficiently after reaching the steady state. Resulting to improve the performance and the smoothness of the driving system. To perform embedded system's deployment, I applied and analyzed different implementations of the proposed method from source utilization point of view. The implementations included

logical optimization, placement of logic cells, and routing the connections between cells.
Related Publications: [RBV20], [RV20], [BRV20]

Thesis II
I proposed deep neural network-based control strategy as an alternative solution to the classical MPC controller for automated driving task aiming to reduce the complexity of solving the online-optimization problem, therefore the execution time. I designed and trained the DNN-based controller to imitate the behaviour of the traditional MPC controller. I also proposed a new automatic intellectual property generator tool, which is developed not only to perform but also to optimize the deployments of deep neural networks on low-end Field FPGA.
Related Publications: [RBV21], [KRVB20a], [ch5]

Thesis III
I proposed machine learning-based control strategy that combines supervised learning (DNN model) and reinforcement learning (RL-model) algorithms in one controller, aiming to achieve the optimization by leveraging the advantages of these algorithms in a way that the RL controller optimizes the actions that are taken by the supervised DNN controller. I evaluated the efficiency of the hybrid model compared to the supervised DNN and reinforcement DQN models which are developed for the same task. The combined model provided the best result and achieved the expected optimization by outputting accurate control actions which reduced the overshooting behaviour, resulting a significant improvement in terms of the smoothness of the driving system. **Related Publications: [ch623], [KRVB20b], [RBV21]**

Thesis IV
I proposed reinforcement learning-based control strategy for the task of maintaining a safe distance in the frame of automated driving system. I also proposed a method to deploy the developed model on low-end FPGA. The method extracts the policy of the trained RL-agent and converts it to a C code that is downloaded and run of the target SoC (FPGA). Compared to the traditional MPC controller, the results show the superiority of the reinforcement learning - based model in term of the ability to predict and follow the changes in the environment state and improvement in response time (1,75 s in average). This work contributed to enriching the research on RL algorithms and paving the way to bring it closer to real-world.
Related Publications: [RV23]

Appendix A: List of Publications

1. Design and implementation of reinforcement learning for automated driving compared to classical MPC control [RV23].
2. Model Predictive Control for Autonomous Quadrotor Trajectory Tracking [BRV20].
3. Mély neurális háló modul generator [VRBD22].
4. Deep Learning-Based Automated Vehicle Steering [RBV21].
5. Survey on five nature-inspired optimization algorithms [RZ21].
6. Model-Based Control Strategy for Autonomous Vehicle Path Tracking Task [RV20].
7. A Survey about Intelligent Solutions for Autonomous Vehicles based on FPGA [KRVB20b].
8. FPGA-based Intelligent Solutions for Autonomous Vehicles: A Short Survey [KRVB20a].
9. Model Predictive Control for Automated Vehicle Steering [RBV20].
10. Toward an embedded system for gesture recognition based on artificial neural network using reconfigurable target (case study and review) [RABV20].
11. A Hybrid Machine Learning-Based Control Strategy for Autonomous Driving Optimization [ch623].
12. Model Predictive-Based DNN Control Model for Automated Steering Deployed on FPGA Using an Automatic IP Generator Tool [ch5].

Bibliography

[AB09] Alessandro Alessio and Alberto Bemporad. Nonlinear model predictive control. *Control and Information Sciences*, 384(2009):345–369, 2009.

[ADBB17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[AG11] Veronica Adetola and Martin Guay. Robust adaptive mpc for constrained uncertain nonlinear systems. *International Journal of Adaptive Control and Signal Processing*, 25(2):155–167, 2011.

[AGT14] Aaron D. Ames, Jessy W. Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs with application to adaptive cruise control. In *IEEE 53rd Annual Conference on Decision and Control*, pages 6271–6278. IEEE, 2014.

[Ahm20] Hossam O. Ahmed. 25.3 gops autonomous landing guidance assistant system using systolic fuzzy logic system for urban air mobility (uam) vehicles using fpga. In *2020 Integrated Communications Navigation and Surveillance Conference (ICNS)*, pages 5D2–1–5D2–11. IEEE, 2020.

[AMM$^+$18] Esraa Adel, Rana Magdy, Sara Mohamed, Mona Mamdouh, Eman El Mandouh, and Hassan Mostafa. Accelerating deep neural networks using fpga. In *2018 30th International Conference on Microelectronics (ICM)*, pages 176–179. IEEE, 2018.

[AOS05] De Luca Alessandro, Giuseppe Oriolo, , and Claude Samson. *Robot Motion Planning and Control*. Springer, Berlin, Heidelberg, Germany, 2005.

[Arb60] ed Arbib, Michael A. *The handbook of brain theory and neural networks*. MIT Press, 1960.

[AW18] Jahanzeb Ahmad and Alexander Warren. Fpga based deterministic latency image acquisition and processing system for automated driving systems. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[AZHK17] Noor Hafizah Amer, Hairi Zamzuri, Khisbullah Hudha, and Zulkiffli Abdul Kadir. Modelling and control strategies in path tracking control for autonomous ground vehicles: a review of state of the art and challenges. *Journal of intelligent & robotic systems*, 86(2):225–254, 2017.

[BFEG17] Matthew Brown, Joseph Funke, Stephen Erlien, and J. Christian Gerdes. Safe driving envelopes for path tracking in autonomous vehicles. *Rocedia-Social and Behavioral Sciences*, (61):307–316, 2017.

[BGC$^+$21] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Paixão, Filipe Mutz, Lucas de Paula Veronese, Thiago Oliveira Santos, and Alberto De Souza. Self-driving cars: A survey. *Expert Systems with Application*, 165(1), 2021.

[BGR04] Vibhor L. Bageshwar, William L. Garrard, and Rajesh Rajamani. Model predictive control of transitional maneuvers for adaptive cruise control vehicles. *IEEE Transactions on Vehicular Technology*, 53(5):1573–1585, 2004.

[BH18] Yecheng Lyu; Lin Bai and Xinming Huang. Real-time road segmentation using lidar data processing on an fpga. In *2018 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2018.

[Bin18] Hiroki Bingo. Development of a control target recognition for autonomous vehicle using fpga with python. In *2018 International Conference on Field-Programmable Technology (FPT)*, pages 419–420. IEEE, 2018.

[BMKY18] Michael G. Bechtel, Elise Mcellhiney, Minje Kim, and Heechul Yun. Deeppicar: A low-cost deep neural network-based autonomous car. In *2018 IEEE 24th international conference on embedded and real-time computing systems and applications (RTCSA)*, pages 11–21. IEEE, 2018.

[BRV20] Rabab Benotsmane, Ahmad Reda, and Jozsef Vasarhelyi. Model predictive control for autonomous quadrotor trajectory tracking. In *2022 23rd International Carpathian Control Conference (ICCC)*, pages 215–220, 2020.

[BSBG12] Assia Belbachir, Jean-Christophe Smal, Jean-Marc Blosseville, and Dominique Gruyer. Simulation-driven validation of advanced driving-assistance systems. *Procedia-Social and Behavioral Sciences*, (48):1205–1214, 2012.

[BSDD17] Andrew Best, Narang Sahil, Barber Daniel, and Manocha Dinesh. Autonovi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2629–2636. IEEE, 2017.

[BVP$^+$11] Hoseinali Borhan, Ardalan Vahidi, Anthony M. Phillips, Ming L. Kuang; Ilya V. Kolmanovsky, and Stefano Di Cairano. Mpc-based energy management of a power-split hybrid electric vehicle. *IEEE Transactions on Control Systems Technology*, 20(3):593–603, 2011.

[CDKDD21] Ian Colbert, Jake Daly, Ken Kreutz-Delgado, and Srinjoy Das. A competitive edge: Can fpgas beat gpus at dcnn inference acceleration in resource-limited edge computing applications. *arXiv*, 2102(00294), 2021.

[CEES14] Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz, and Robert W. Stewart. *The Zynq book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. Strathclyde Academic Media, 2014.

[ch212] Adaptive dynamic surface control for formations of autonomous surface vehicles with uncertain dynamics. *IEEE Transactions on Control Systems Technology*, 21(2):513–520, 2012.

[ch5] Model predictive-based dnn control model for automated steering deployed on fpga using an automatic ip generator tool. (220).

[ch623] A hybrid machine learning-based control strategy for autonomous driving optimization. (1):–, 2023.

[CHPB21] Walther Carballo-Hernández, Maxime Pelcat, and François Berry. "why is fpga-gpu heterogeneity the best option for embedded deep neural networks? *arXiv*, 2102(01343), 2021.

[CK18] Stefano Di Cairano and Ilya V. Kolmanovsky. Real-time optimization and model predictive control for aerospace and automotive applications. In *2018 annual American control conference (ACC)*, pages 2392–2409. IEEE, 2018.

[CKK$^+$17] Hyunmin Chae, Chang Mook Kang, ByeoungDo Kim, Jaekyum Kim, Chung Choo Chung, and Jun Won Choi. Autonomous braking system via deep reinforcement learning. In *20th International conference on intelligent transportation systems (ITSC)*, pages 1–6. IEEE, 2017.

[CLG$^+$17] Wang Chao, Wenqi Lou, Lei Gong, Lihui Jin, Luchao Tan, Yahui Hu, Xi Li, and Xuehai Zhou. Reconfigurable hardware accelerators: Opportunities, trends, and challenges. *arXiv*, 1712(04771), 2017.

[CLSH06] Danielem Corona, Mircea Lazar, Bart De Schutter, and Maurice Heemels. A hybrid mpc approach to the design of a smart adaptive cruise controller. In *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, pages 13–15. IEEE, 2006.

[Com14] SAE On-Road Automated Vehicle Standards Committee. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. In *Technical Report*, pages 1–6. 2014.

[CQCD15] Katrakazas Christosand, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, (60):416–442, 2015.

[DCD11] Charles Desjardins and Brahim Chaib-Draa. Cooperative adaptive cruise control: A reinforcement learning approach. *IEEE Transactions on intelligent transportation systems*, 12(4):1248–1260, 2011.

[DHa$^+$19] Sen Du, Tian Huang, Junjie Hou a, Shijin Song, and Yuefeng Song. Fpga based acceleration of game theory algorithm in edge computing for autonomous driving. *Journal of Systems Architecture*, (93):33–39, 2019.

[DLH$^+$13] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, Yifan Gong, and Alex Acero. Recent advances in deep learning for speech research at microsoft. In *13 IEEE international conference on acoustics, speech and signal processing*, pages 8604–08608. IEEE, 2013.

[DLTR17] Mathieu Deremetz, Roland Lenain, Benoit Thuilot, and Vincent Rousseau. Adaptive trajectory control of off-road mobile robots: A multi-model observer approach. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4407–4413. IEEE, 2017.

[DM20] Maha S. Diab and Soliman A. Mahmoud. Survey on field programmable analog array architectures eliminating routing network. *IEEE Access*, (8):220779–220794, 2020.

[DTMD08] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *American Associate for Artificial Intelligence*, 1001(48105):18–80, 2008.

[DTP21] George Dimitrakopoulos, Aggelos Tsakanikas, and Elias Panagiotopoulos. A path of structural transformation for the automotive and insurance industries toward autonomous vehicles. In *Autonomous Vehicles*, pages 69–83. Elsevier, 2021.

[DYW$^+$15] Kakan C. Dey, Li Yan, Xujie Wang, Yue Wang, Haiying Shen, Mashrur Chowdhury, Chenxi Qiu Lei Yu, and Vivekgautham Soundararaj. A review of communication, driver characteristics, and controls aspects of cooperative adaptive cruise control (cacc). *IEEE Transactions on Intelligent Transportation Systems*, 17(2):491–509, 2015.

[DZ87] Ernst D. Dickmanns and Alfred Zapp. Autonomous high speed road vehicle guidance by computer vision. *Electronics*, 5(12):221–226, 1987.

[EDC14] Matthew Ellis, Helen Durand, and Panagiotis D. Christofides. A tutorial review of economic model predictive control methods. *Journal of Process Control*, 24(8):1156–1178, 2014.

[EGCW18] Damien Ernst, Mevludin Glavic, Florin Capitanescu, and Louis Wehenkel. Reinforcement learning versus model predictive control: a comparison on a power system problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):517–529, 2018.

[Eig17] György Eigner. Control of physiological systems through linear parameter varying framework. *Acta Polytechnica Hungarica*, 14(6):185–212, 2017.

[FLVH+18] François-Lavet, Vincent, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3–4):219–354, 2018.

[FNCMBF08] Wanderley C. Felipe N., Martinsand Celeste, Ricardo Carelli, Sarcinelli-Filho Mário, and Teodiano F. Bastos-Filho. An adaptive dynamic controller for autonomous mobile robot trajectory tracking. *Control Engineering Practice*, 16(11):1354–1363, 2008.

[FTAH08] Paolo Falcone, H. Eric Tseng, Francesco Borrelliand Jahan Asgari, and Davor Hrovat. Mpc-based yaw and lateral stabilisation via active front steering and braking. *Vehicle System Dynamics*, 46(s1):611–628, 2008.

[GDDM14a] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[GDDM14b] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587. IEEE, 2014.

[GLL17] Shixianga nd Ethan Holly Gu, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[GLX+17] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 152–159. IEEE, 2017.

[GS10] Jorge L. Garriga and Masoud Soroush. Model predictive control tuning methods: A review. *Industrial and Engineering Chemistry Research*, 49(8):3505–3515, 2010.

[GS20] Alireza Ghaffari and Yvon Savaria. Cnn2gate: An implementation of convolutional neural networks inference on fpgas with automated design space exploration. *Electronics*, 9(12):2200, 2020.

[HAM18] Rihab Hmida, Abdessalem Ben Abdelali, and Abdellatif Mtibaa. Hardware implementation and validation of a traffic road sign detection and identification system. *Journal of Real-Time Image Processing*, (15):13–30, 2018.

[HGCV21] Salim Hima, Sebastien Glaser, Ahmed Chaibet, and Benoit Vanholme. Controller design for trajectory tracking of autonomous passenger vehicles. In *2011 14th international IEEE conference on intelligent transportation systems (ITSC)*, pages 1459–1464. IEEE, 2021.

[HKP$^+$11] Tamás Haidegger, Levente Kovács, Radu-Emil Precup, Stefan Preitl, Balázs Benyó, and Zoltán Benyó. Cascade control for telerobotic systems serving space medicine. *FAC Proceedings Volumes*, 44(1):3759–3764, 2011.

[How60] Ronald A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, USA, 1960.

[HSJ$^+$19] Cong Hao, Atif Sarwari, Zhijie Jin, Husam Abu-Haimed, Daryl Sew, Yuhong Li, Xinheng Liu, Bryan Wu, Dongdong Fu, Junli Gu, and Deming Chen. A hybrid gpu+ fpga system design for autonomous driving cars. In *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 121–126. IEEE, 2019.

[HTMT07] Gabriel M. Hoffmann, Claire J. Tomlin, Michael Montemerlo, and Sebastian Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *2007 American control conference*, pages 2296–2301. IEEE, 2007.

[HVPO15] Yan Han, Kushal Virupakshappa, Esdras Vitor Silva Pinto, and Erdal Oruklu. Hardware/software co-design of a traffic sign recognition system using zynq fpgas. *Electronics*, 4(4):1062–1089, 2015.

[HWA19] Koki Honda, Kaijie Wei, and Hideharu Amano. Fpga/python co-design for lane line detection on a pynq-z1 board. In *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 53–60. IEEE, 2019.

[HXXTS17] Zhenhua Huang, Haibo He Xin Xu, Jun Tan, and Zhenping Sun. Parameterized batch reinforcement learning for longitudinal control of autonomous land vehicles. *Transactions on Systems, Man, and Cybernetics: Systems*, 49(4):730–741, 2017.

[IAZB08] Md Shabiul Islam, Nowshad Amin, Mukter Zaman, and MS. Bhuyan. Fuzzy based pid controller using vhdl for transportation application. *International Journal of Mathematical Models and Methods in Applied Sciences*, 2(2):143–147, 2008.

[IRC$^+$18] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. avigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 2034–2039. IEEE, 2018.

[IWC07] Petros Ioannou, Yun Wang, and Hwan Chang. Integrated roadway/adaptive cruise control system: Safety, performance, environmental and near term deployment considerations. In *California PATH Research Report UCB-ITS-PRR,* *Institute of Transportation Studies, University of California: Berkeley, USA*. 2007.

[JBCG19] Junior C. Jesus, Jair A. Bottega, Marco ASL Cuadros, and Daniel FT Gamarra. Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 362–367. IEEE, 2019.

[JGCC15] Wingbermuehle Joseph G., Ron K. Cytron, and Roger D. Chamberlain. Superoptimized memory subsystems for streaming applications. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 126–135, 2015.

[JSS09] Wang Junmin, Joe Steiber, and Bapiraju Surampudi. Autonomous ground vehicle control system for high-speed and safe operation. *International Journal of Vehicle Autonomous Systems*, 7(1-2):18–35, 2009.

[JUG15] Dørum Jarle, Tryve Utstumo, and Jan Tommy Gravdahl. Experimental comparison of adaptive controllers for trajectory tracking in agricultural robotics. In *2015 19th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 206–212. IEEE, 2015.

[KAE$^+$12] Roozbeh Kianfar, Bruno Augusto, Alireza Ebadighajari, Usman Hakeem, Josef Nilsson, Ali Raza, Reza S. Tabar, Naga VishnuKanth Irukulapati, Paolo Falcone, Stylianos Papanastasiou, Lennart Svensson, and Henk Wymeersch. Design and experimental validation of a cooperative driving system in the grand cooperative driving challenge. *IEEE transactions on intelligent transportation systems*, 13(3):994–1007, 2012.

[KBJ$^+$20] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):712–733, 2020.

[KLKK20] Syed Ans Bin Khalid, Eyke Liegmann, Petros Karamanakos, and Ralph Kennel. High-level synthesis of a long horizon model predictive control algorithm for an fpga. In *International Exhibition and Conference for Power Electronics, Intelligent Motion, Renewable Energy and Energy Management*, pages 1–8, 2020.

[KRVB20a] Ashraf Kasem, Ahmad Reda, József Vásárhelyi, and Ahmed Bouzid. Fpga-based intelligent solutions for autonomous vehicles: A short survey. In *The 1st Conference on Information Technology and Data Science*, pages 94–96, 2020.

[KRVB20b] Ashraf Kasem, Ahmad Reda, József Vásárhelyi, and Ahmed Bouzid. A survey about intelligent solutions for autonomous vehicles based on fpga. *CARPATHIAN JOURNAL OF ELECTRONIC AND COMPUTER ENGINEERING*, 13(2):7–11, 2020.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[KTGL18] Steven Spielberg Pon Kumar, Aditya Tulsyan, Bhushan Gopaluni, and Philip Loewen. A deep learning architecture for predictive control. *IFAC-PapersOnLine*, 51(18):512–517, 2018.

[KVRB20] Alexandros Kouris, Stylianos I. Venieris, Michail Rizakis, and Christos-Savvas Bouganis. Approximate lstms for time-constrained inference: Enabling fast reaction in self-driving cars. *IEEE Consumer Electronics Magazine*, 9(4):11–26, 2020.

[KZN19] Fülöp Kóta, Tamás Zsedrovits, and Zoltán Nagy. Sense-and-avoid system development on an fpga. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 575–579. IEEE, 2019.

[LBH18] Yecheng Lyu, Lin Bai, and Xinming Huang. Real-time road segmentation using lidar data processing on an fpga. In *2018 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2018.

[LC19] Changjian Li and Krzysztof Czarnecki. Urban driving with multi-objective deep reinforcement learning. In *International Foundation for Autonomous Agents and Multiagent Systems (2019)*, pages 359–367, 2019.

[LHP$^+$15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Tom Erez Nicolas Heess, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint*, 1509(02971), 2015.

[LHW20] Jeonghun Lee, Jiayuan He, and Ke Wang. Fpga-based neural network accelerators for millimeter-wave radio-over-fiber systems. *Optics Express*, 28(9):13384–13400, 2020.

[Li17] Yuxi Li. Deep reinforcement learning: An overview. *arXiv*, 1701(07274), 2017.

[LLL$^+$19] Yilan Li, Hongjia Li, Zhe Li, Haowen Fang, Amit K. Sanyal, Yanzhi Wang, and Qinru Qiu. Fast and accurate trajectory tracking for unmanned aerial vehicles based on deep reinforcement learning. In *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–9. IEEE, 2019.

[LOG$^+$18] Xu Liu, Hibat Allah Ounifi, Abdelouahed Gherbi, Yves Lemieux, and Wubin Li b. A hybrid gpu-fpga-based computing platform for machine learning. *Procedia Computer Science*, (141):104–111, 2018.

[LWM08] K.V. Ling, B.F. Wu, and J.M. Maciejowski. Embedded model predictive control (mpc) using a fpga. *The International Federation of Automatic Control*, 41(2):15250–15255, 2008.

[LWZC15] Xiang Li, Zhuping Wang, Jin Zhu, and Qijun Chen. Adaptive tracking control for wheeled mobile robots with unknown skidding. In *2015 IEEE conference on control applications (CCA)*, pages 1674–1679. IEEE, 2015.

[LXL17] Qian Li, Qingcheng Xiao, and Yun Liang. Enabling high performance deep learning networks on embedded systems. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 8405–8410. IEEE, 2017.

[LYLM09] Mark S. K. Lau, S. P. Yue, K. V. Ling, and J. M. Maciejowski. A comparison of interior point and active set methods for fpga implementation of model predictive control. In *2009 European Control Conference (ECC)*, pages 156–161. IEEE, 2009.

[LYM06] K.V. Ling, S.P. Yue, and J.M. Maciejowski. A fpga implementation of model predictive control. In *American Control Conference*, pages 15250–15255. IEEE, 2006.

[LYS18] Imad Lamouik, Ali Yahyaouy, and My Abdelouahed Sabri. Deep neural network dynamic traffic routing system for vehicles. In *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–4. IEEE, 2018.

[LZLS15] Teng Liu, Yuan Zou, Dexing Liu, and Fengchun Sun. Reinforcement learning of adaptive energy management with transition probability for a hybrid electric tracked vehicle. *IEEE Transactions on Industrial Electronics*, 62(12):7837–7846, 2015.

[MA17] Silvia Magdici and Matthias Althoff. Adaptive cruise control with safety guarantees for autonomous vehicles. *IFAC-PapersOnLine*, 50(1):5774–5781, 2017.

[Mah20] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR)*, (9):381–386, 2020.

[Mat] MathWorks. *MPC Modelling*. www.mathworks.com/help/mpc/gs/mpc-modeling.html. visited 01/01/2020.

[Mat18a] MathWorks. Choose sample time and horizons, 2018. visited 16/03/2020.

[Mat18b] MathWorks. Deep learning hdl toolbox, 2018.

[MBBH21] Daniel García Moreno, Alberto A. Del Barrio, Guillermo Botella, and Jennifer Hasler. A cluster of fpaas to recognize images using neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(11):3391–3395, 2021.

[MCGZCM19] Óscar Mata-Carballeira, Jon Gutiérrez-Zaballa, Inés Del Campo, and Victoria Martínez. An fpga-based neuro-fuzzy sensor for personalized driving assistance. *Sensors*, 19(18):4011, 2019.

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis

Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[NG19] Balázs Németh and Péter Gáspár. Lpv design for the control of heterogeneous traffic flow with autonomous vehicles. *Acta Polytechnica Hungarica*, 16(7):233–246, 2019.

[NHC+14] Petter Nilsson, Omar Hussien, Yuxiao Chen, Ayca Balkan, Matthias Rungger, Aaron Ames, Jessy Grizzle, Necmiye Ozay, Huei Peng, and Paulo Tabuada. Preliminary results on correct-by-construction control software synthesis for adaptive cruise control. In *53rd IEEE Conference on Decision and Control*, pages 816–823. IEEE, 2014.

[nHTK15] Deepak ngole, Juraj Holaza, Bálint Takács, and Michal Kvasnica. Fpga-based explicit model predictive control for closed-loop control of intravenous anesthesia. In *2015 20th International Conference on Process Control (PC)*, pages 42–478. IEEE, 2015.

[NHZHAK11] Amer Noor Hafizah, Hairi Zamzuri, Khisbullah Hudha, and Zulkiffli Abdul Kadir. Vehicle dynamic stability improvements through gain-scheduled steering and braking control. *Vehicle System Dynamics*, 49(10):1597–1621, 2011.

[NSMN10] Andre Benine Neto, Stefano Scalzi, Said Mammar, and Mariana Netto. Dynamic controller for lane keeping and obstacle avoidance assistance system. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1363–1368. IEEE, 2010.

[NVR13] Quang Thuan Nguyen, Vojtech Veselý, and Danica Rosinová. Design of robust model predictive control with input constraints. *International Journal of systems science*, 44(5):896–907, 2013.

[OMM+14] Norliza Othman, Farhanahani Mahmud, Abd Kadir Mahamad, M. Hairol Jabbar, and Nur Atiqah Adon. Cardiac excitation modeling: Hdl coder optimization towards fpga stand-alone implementation. In *014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014)*, pages 507–511. IEEE, 2014.

[PBT+08] Falcone Paolo, Francesco Borrelli, H. Eric Tseng, Jahan Asgari, and Davor Hrovat. A hierarchical model predictive control framework for autonomous ground vehicles. In *2008 American Control Conference*, pages 3719–3724. IEEE, 2008.

[PCvL11] Zhao Pan, Jiajia Chen, Tao v, and Huawei Liang. Dynamic motion planning for autonomous vehicle in unknown environments. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 284–289. IEEE, 2011.

[PLH15] Myungwook Park, Sangwoo Lee, and Wooyong Han. Development of steering control system for autonomous vehicle using geometry-based path tracking algorithm. *ETRI Journal*, 37(3):617–625, 2015.

[PSN$^+$17] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 IEEE international conference on robotics and automation (icra)*, pages 1527–1533. IEEE, 2017.

[Put14] Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, USA, 2014.

[QZF$^+$20] Zhiqiang Que, Yongxin Zhu, Hongxiang Fan, Jiuxi Meng, Xinyu Niu, , and Wayne Luk. Mapping large lstms to fpgas with weight reuse. *Journal of Signal Processing Systems*, (92):965–979, 2020.

[RABV20] Ahmad Reda, Tareq Alshoufi, Ahmed Bouzid, and József Vásárhelyi. Toward an embedded system for gesture recognition based on artificial neural network using reconfigurable target (case study and review). *MULTIDISZCIPLINÁRIS TUDOMÁNYOK: A MISKOLCI EGYETEM KÖZLEMÉNYE*, 10(4):142–148, 2020.

[RBV20] Ahmad Reda, Ahmed Bouzid, and József Vásárhelyi. Model predictive control for automated vehicle steering. *ACTA POLYTECHNICA HUNGARICA*, 17(7):163–182, 2020.

[RBV21] Ahmad Reda, Ahmed Bouzid, and Jozsef Vasarhelyi. Deep learning-based automated vehicle steering. In *22nd International Carpathian Control Conference (ICCC 2021)*, pages 1–5, 2021.

[RGNR$^+$09] Guilherme V. Raffo, Guilherme K. Gomes, Julio E Normey-Rico, Christian R. Kelber, and Leandro B. Becker. A predictive controller for autonomous vehicle path tracking. *IEEE transactions on intelligent transportation systems*, 10(1):92–102, 2009.

[RHS$^+$17] Viktor Rausch, Andreas Hansen, Eugen Solowjow, Chang Liu, Edwin Kreuzer, and J. Karl Hedrick. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *2017 American Control Conference (ACC)*, pages 4914–4919. IEEE, 2017.

[RNS04] Siegwart Rol, Illah Reza Nourbakhsh, and Davide Scaramuzza. Introduction to autonomous mobile robots. In *A Bradford Book*. MIT Press, Cambridge, USA, 2004.

[Rot17] Alan Rothwell. Optimization methods in structural design. *Cham: Springer*, 242:55–81, 2017.

[RR21] Rebecca L. Russell and Christopher Reale. Multivariate uncertainty in deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):7937–7943, 2021.

[RSB$^+$05] Gerhard Rieger, Joachim Scheef, Holger Becker, Michael Stanzel, and Robert Zobel. Active safety systems change accident environment of vehicles significantly-a challenge for vehicle design. In *International Technical Conference on the Enhanced Safety of Vehicles (ESV)*, pages 6–9, 2005.

[RV20] Ahmad Reda and József Vásárhelyi. Model-based control strategy for autonomous vehicle path tracking task. *ACTA UNIVERSITATIS SAPIENTIAE ELECTRICAL AND MECHANICAL ENGINEERING*, 12(2020):35–45, 2020.

[RV23] Ahmad Reda and József Vásárhelyi. Design and implementation of reinforcement learning for automated driving compared to classical mpc control. *Designs*, 7(1):1–18, 2023.

[RZ21] Ahmad Reda and Csaba Johanyák Zsolt. Survey on five nature-inspired optimization algorithms. *GRADUS*, 8(1):173–183, 2021.

[SB99] Richard S. Sutton, , and Andrew G. Barto. *Reinforcement learning: An introduction.* MIT Press, Cambridge, USA, 1999.

[SBS13] David Silver, J. Andrew Bagnell, , and Anthony Stentz. Learning autonomous driving styles and maneuvers from expert demonstration. In *13th International Symposium on Experimental Robotics*, pages 371–386. Springer, 2013.

[SdR13] Thomas Stanger and Luigi del Re. A model predictive cooperative adaptive cruise control approach. In *Proceedings of the American Control Conference*, pages 1374–1379. IEEE, 2013.

[SEZEHE20] Asmaa Swief, Amr El-Zawawi, Mohamed El-Habrouk, and Amr Nasr Eldin. Approximate neural network model for adaptive model predictive control. In *2020 16th International Computer Engineering Conference (ICENCO)*, pages 135–140. IEEE, 2020.

[SGH15] Fynn Schwiegelshohn, Lars Gierke, and Michael Hübner. Fpga based traffic sign detection for automotive camera systems. In *2015 10th international symposium on reconfigurable communication-centric systems-on-chip (ReCoSoC)*, pages 1–6. IEEE, 2015.

[SH17] Sahil Shah and Jennifer Hasler. oc fpaa hardware implementation of a vmm+ wta embedded learning classifier. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):28–37, 2017.

[SKM$^+$19] Christ Freben Dommaris Saragih, Fabiola Maria Teresa Retno Kinasih, Carmadi Machbub, Pranoto Hidaya Rusmin, and Arief Syaichu Rohman. Visual servo application using model predictive control (mpc) method on pan-tilt camera platform. In *2019 6th International Conference on Instrumentation, Control*, pages 1–7. IEEE, 2019.

[SLH$^+$14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[SMSM99] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, (12):1057–1063, 1999.

[Sni09] Jarrod M. Snider. Automatic steering methods for autonomous automobile path tracking. *in Robotics Institute, Carnegie Mellon University*, USA, 2009.

[sOmY19] Chang song Oh and Jong min Yoon. Hardware acceleration technology for deep-learning in autonomous vehicles. In *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–3. IEEE, 2019.

[ST13] Yam P. Siwakoti and Graham E. Town. Design of fpga-controlled power electronics and drives using matlab simulink. In *2013 IEEE ECCE Asia Downunder*, pages 571–577. IEEE, 2013.

[STAK18] Per Skarin, William Tärneberg, Karl-Erik Arzen, and Maria Kihl. Towards mission-critical control at the edge and over 5g. In *2018 IEEE international conference on edge computing (EDGE)*, pages 50–57. IEEE, 2018.

[STH20] Vijay Kumar Singh, Ravi Nath Tripathi, and Tsuyoshi Hanamoto. Implementation strategy for resource optimization of fpga-based adaptive finite control set-mpc using xsg for a vsi system. *IEEE Journal of Emerging and Selected Topics in Power Electronics*, 9(2):2066–2078, 2020.

[TKP$^+$11] Haidegger Tamas, Levente Kovács, Stefan Preitland Radu-Emil Precup, Balázs Benyó, and Zoltán Benyó. Controller design solutions for long distance telesurgical applications. *International Journal of Artificial Intelligence*, 6(S11):48–71, 2011.

[TMD$^+$06] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Mark Oakley, Celia Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, , Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[TMT06] S.E. Tuna, M.J. Messina, and A.R. Teel. Shorter horizons for model predictive control. In *2006 American Control Conference*, pages 863–868. IEEE, 2006.

[UDoT18] USA U.S Department of Transportation. *Preparing for the Future of Transportation: Automated Vehicles 3.0.* 2018.

[Ven10] Gerhard Venter. *Review of Optimization Techniques.* John Wiley & Sons, Chichester, UK, 2010.

[VRBD22] József Vásárhelyi, Ahmad Reda, Ahmed Bouzid, and Daniel Drótos. Mély neurális háló modul generátor. In *Sebestyén-Pál, György; Szabó, Loránd (szerk.) ENELKO 2022, XXIII. Energetika-Elektrotechnika – ENELKO és XXXII. Számítástechnika és Oktatás – SzámOkt Multi-konferencia*, pages 191–195., 5 p., 2022.

[VSFA14] Kizheppatt Vipin, Shanker Shreejith, Suhaib A. Fahmy, and Easwaran Arvind. Mapping time-critical safety-critical cyber physical systems to hybrid fpgas. In *2014 IEEE International Conference on Cyber-Physical Systems, Networks, and Applications*, pages 31–36. IEEE, 2014.

[WCdLF18] Pin Wang, Ching-Yao Chan, and Arnaud de La Fortelle. A reinforcement learning based approach for automated lane change maneuvers. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1379–1384. IEEE, 2018.

[WGDL21] Ran Wu, Xinmin Guo, Jian Du, and Junbao Li. Accelerating neural network inference on fpga-based platforms—a survey. *Electronics*, 10(9):1025, 2021.

[WHA18] Kaijie Wei, Koki Honda, and Hideharu Amano. Fpga design for autonomous vehicle driving using binarized neural networks. In *International Conference on Field-Programmable Technology (FPT)*, pages 428–431. IEEE, 2018.

[WIP02] USA Wiley-IEEE Press, New York. *Linear Time-Invariant Systems.* 2002.

[WKA+18] Hiromichi Wakatsuki, Takao Kido, Kenta Arai, Yuhei Sugata, Kanemitsu Ootsu, Takashi Yokota, and Takeshi Ohkawa. Development of a robot car by single line search method for white line detection with fpga. In *2018 International Conference on Field-Programmable Technology (FPT)*, pages 415–418. IEEE, 2018.

[WLJ19] Tianze Wu, Weiyi Liu, and Yongwei Jin. end-to-end solution to autonomous driving based on xilinx fpga. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 427–430. IEEE, 2019.

[WLTY08] Xing Wu, Peihuang Lou, Dunbing Tang, and Jun Yu. An intelligent-optimal predictive controller for path tracking of vision-based automated guided vehicle. In *2008 International Conference on Information and Automation*, pages 844–849. IEEE, 2008.

[Wor12] Karl Worthmann. Estimates of the prediction horizon length in mpc: A numerical case study. *IFAC Proceedings Volumes*, 45(17):232–237, 2012.

[WZZ+18] Shouyang Wei, Yuan Zou, Tao Zhang, Xudong Zhang, and Wenwei Wang. Design and experimental validation of a cooperative adaptive cruise control system based on supervised reinforcement learning. *Applied sciences*, 8(7):1014, 2018.

[XHL+18] Ji Xuewu, Xiangkun He, Chen Lv, Yahui Liu, and Jian Wu. Adaptive-neural-network-based robust lateral motion control for autonomous vehicle at driving limits. *Control Engineering Practice*, (76):41–53, 2018.

[XIL] XILINX. *MicroBlaze.* https://www.xilinx.com/products/intellectual-property/microblazecore.html. visited 08/04/2021.

[Xil16] Xilinx. Vivado design suite user guide: Implementation, ug904, v2016.2, 2016.

[XIL20] XILINX. *White Paper: Versal ACAPs.* WP505 (v1.1.1), 2020.

[YGDWS13] Xi Yu-Geng, Li De-Wei, and Lin Shu. Model predictive control—status and challenges. *Acta Automatica Sinica*, 39(3):222–236, 2013.

[YLCT20] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, (20):58443–58469, 2020.

[YLLL17] Xiaoliang Yang, Guorong Liu, Anping Li, and Van Dai Le. A predictive power control strategy for dfigs based on a wind energy converter system. *Energies*, 10(8):1098, 2017.

[YZO16] Andre Shigueo Yamashita, Antônio Carlos Zanin, , and Darci Odloak. Tuning of model predictive control with multi-objective optimization. *Brazilian Journal of Chemical Engineering*, 33(2):333–346, 2016.

[ZC13] Lin Zhongduo and Paul Chow. A zynq-based hadoop cluster. In *2013 International Conference on Field-Programmable Technology (FPT)*, pages 450–453, 2013.

[ZCGC17] Adam Ziebinski, Rafal Cupek, Damian Grzechca, and Lukas Chruszczyk. Review of advanced driver assistance systems (adas). In *AIP Conference Proceedings*, pages 120002–1– 120002–4. AIP Publishing LLC, 2017.

[ZCN$^+$12] Sun Zhenping, Qingyang Chen, Yiming Nie, Daxue Liu, and Hangen He. Ribbon model based path tracking method for autonomous land vehicle. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1220–1226. IEEE, 2012.

[ZWZ$^+$18] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen mei Hwu, and Deming Chen. Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD*, pages 1–8. IEEE, 2018.