National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

Prediction of Vehicle Behavior in Urban Environments Using Machine Learning

Diploma Thesis

of

SPYRIDON PARASKEVAS STENTOUMIS

Supervisor: Panagiotis Tsanakas
Professor, NTUA

Athens, October 2025

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

Prediction of Vehicle Behavior in Urban Environments Using Machine Learning

Diploma Thesis

of

SPYRIDON PARASKEVAS STENTOUMIS

Supervisor: Panagiotis Tsanakas
Professor, NTUA

Approved by the examination committee on 30th October 2025.

(Signature)	*(Signature)*	*(Signature)*
Panagiotis Tsanakas	Dimitrios Sountris	Andreas Georgios Stafylopatis
Professor, NTUA	Professor, NTUA	Professor, NTUA

Athens, October 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Πρόβλεψη συμπεριφοράς οχημάτων σε περιβάλλον πόλης με χρήση Μηχανικής Μάθησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΠΥΡΙΔΩΝ ΠΑΡΑΣΚΕΥΑ ΣΤΕΝΤΟΥΜΗ

Επιβλέπων : Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Οκτωβρίου 2025.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Παναγιώτης Τσανάκας	Δημήτριος Σούντρης	Ανδρέας Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2025

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

(Signature)

. . . . . .

Spyridon Paraskevas Stentoumis
Graduate of Electrical and Computer Engineering, NTUA

30th October 2025

Περίληψη

Η αυτόνομη οδήγηση αποτελεί τον κύριο στόχο πολλών κατασκευαστών οχημάτων σήμερα. Σε ένα πλήρως αυτόνομο σύστημα, το όχημα πρέπει να μπορεί να κινείται χωρίς ανθρώπινη παρέμβαση, γεγονός που απαιτεί πλήρη κατανόηση του περιβάλλοντός του, ακριβή έλεγχο του οχήματος και ικανότητα μετακίνησης από ένα σημείο σε άλλο. Μία από τις κύριες προκλήσεις στην οδήγηση σε αυτοκινητόδρομους, ιδιαίτερα σε δρόμους με πολλές λωρίδες, είναι η πρόβλεψη της αλλαγής λωρίδας από τα γύρω οχήματα. Παρόλο που τα φλας έχουν σχεδιαστεί για να υποδεικνύουν τέτοιους ελιγμούς, οι οδηγοί δεν τα χρησιμοποιούν πάντα με συνέπεια. Ως εκ τούτου, τα αυτόνομα οχήματα πρέπει να είναι σε θέση να ανιχνεύουν και να προβλέπουν αλλαγές λωρίδας βασιζόμενα σε άλλα στοιχεία, ανεξάρτητα από οπτικά σήματα.

Η παρούσα διπλωματική εργασία ασχολείται με το πρόβλημα της πρόβλεψης επικείμενων αλλαγών λωρίδας χρησιμοποιώντας ένα μοντέλο μηχανικής μάθησης βασισμένο σε Δίκτυα Μακροπρόθεσμης Βραχυπρόθεσμης Μνήμης (Long Short-Term Memory, LSTM). Το μοντέλο ταξινομεί παράθυρα σταθερού μήκους σε μία από τρεις κατηγορίες: LLC (Αλλαγή προς Αριστερή Λωρίδα), RLC (Αλλαγή προς Δεξιά Λωρίδα) ή NLC (Καμία Αλλαγή Λωρίδας). Η εργασία παρουσιάζει δύο βασικές καινοτομίες που δεν έχουν διερευνηθεί προηγουμένως στη βιβλιογραφία. Πρώτον, χρησιμοποιεί κινηματικά χαρακτηριστικά για την πρόβλεψη ελιγμών αντί για τη θέση του οχήματος σε τοπικό καρτεσιανό σύστημα ή τις εντολές ελέγχου του οδηγού, οι οποίες συνήθως δεν είναι διαθέσιμες σε πραγματικά σενάρια χωρίς επικοινωνία ″2″. Δεύτερον, αντιμετωπίζει την έλλειψη μεγάλου όγκου διαθέσιμων δημόσιων συνόλων δεδομένων ,δημιουργώντας συνθετικά δεδομένα με τη χρήση του προσομοιωτή ανοικτού κώδικα CARLA, τα οποία στη συνέχεια χρησιμοποιούνται για την εκπαίδευση του μοντέλου. Τέλος, το μοντέλο αξιολογείται (α) σε συνθετικά δεδομένα που παράγονται υπό διαφορετικές συνθήκες και (β) σε πραγματικά σύνολα δεδομένων, αποδεικνύοντας την ικανότητά του να γενικεύει σε διαφορετικά περιβάλλοντα.

Λέξεις Κλειδιά

Αυτόνομη Οδήγηση, Μηχανική Μάθηση, Πρόβλεψη αλλαγής λωρίδας, Ταξινόμηση ελιγμού, LSTM (Long Short-Term Memory), Προσομοιωτής CARLA, Συνθετικά δεδομένα

Abstract

Autonomous driving is the primary objective for many car manufacturers today. In a fully autonomous system, a vehicle must navigate without human input, which requires a comprehensive understanding of its environment, precise control over the vehicle, and the ability to travel from one location to another. A key challenge in highway driving, particularly on multi-lane roads, is anticipating the lane-changing behavior of nearby vehicles. While turn signals are designed to indicate such maneuvers, drivers do not always use them reliably. As a result, autonomous vehicles need to detect and predict lane changes based on other cues, independent of visual signals.

This thesis addresses the problem of predicting imminent lane changes using a machine learning model based on Long Short-Term Memory (LSTM) networks. The model classifies fixed-length sliding windows into one of three categories: LLC (Left Lane Change), RLC (Right Lane Change), or NLC (No Lane Change). The work introduces two key novelties not previously explored in the literature. First, it employs kinematic features for maneuver prediction rather than relying on the vehicle's position in a local Cartesian system or the driver's control inputs, which are generally unavailable in real-world scenarios without V2V communication. Second, it addresses the lack of publicly available large datasets by generating synthetic data using the open-source CARLA Simulator, which are then used to train the model. Finally, the model is evaluated on (a) synthetic data produced under varying conditions, and (b) real-world datasets, demonstrating its capability to generalize across different environments.

Keywords

Autonomous Driving, Machine learning, Lane change prediction, Maneuver classification, LSTM (Long Short-Term Memory), CARLA simulator, Synthetic data

to my family

Ευχαριστίες

Θα ήθελα να εκφράσω τις ευχαριστίες μου προς τον καθηγητή κ. Τσανάκα για την ευκαιρία που μου προσέφερε να εκπονήσω την παρούσα διπλωματική εργασία σε ένα ιδιαίτερα ενδιαφέρον και σύγχρονο αντικείμενο, το οποίο συνδυάζει κρίσιμα ζητήματα, όπως η Μηχανική Μάθηση και η αυτόνομη οδήγηση.

Ευχαριστώ επίσης θερμά τον Δρ. Δραϊνάκη Γεώργιο για την κατανόηση που επέδειξε σχετικά με το απαιτητικό πρόγραμμά μου και τις διάφορες υποχρεώσεις που καθυστέρησαν την ολοκλήρωση της εργασίας, καθώς και για την ευελιξία που μου παρείχε ώστε να διαθέσω τον απαραίτητο χρόνο. Αντίστοιχα, θα ήθελα να ευχαριστήσω και τον κ. Γιώργο Χατζηπαυλή, ο οποίος με βοήθησε στην εγκατάσταση και στη χρήση του προσομοιωτή CARLA και αφιέρωσε χρόνο για να μπορούν να τρέξουν όλα σε Windows που χρησιμοποιούσα εγώ όταν είχαν σχεδιαστεί για Linux.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου, τους φίλους μου και τους συναδέλφους μου όλα τα χρόνια της φοίτησής μου, για τη στήριξή τους τόσο στις δύσκολες όσο και στις ευχάριστες στιγμές, μέσα από τις οποίες αποκόμισα πολύτιμες εμπειρίες.

Αθήνα, Οκτωβρίου 2025

Σπυρίδων Παρασκευάς Στεντούμης

Contents

List of Figures

List of Tables

Preface

This thesis was conducted in close collaboration with Dr. Georgios Drainakis and Mr. Georgios Chatzipavlis, researchers at the National Technical University of Athens, under the supervision of Professor Panagiotis Tsanakis, Dean of the School of Electrical and Computer Engineering.

The preparation of this thesis did not unfold as smoothly as I would have wished, despite my excellent collaboration with Dr. Drainakis, the researcher with whom I worked closely and received guidance, and Mr. Giorgos Chatzipavlis, who helped me with the installation and use of CARLA. The reasons for these difficulties were external factors beyond my control, including my full-time job as a Software Developer and, later, my compulsory military service in the Hellenic Army from late 2024 until mid-2025.

Another challenge in the early stages of this work was my inability to be present in the laboratory, since my working hours coincided with the lab's schedule. As a result, the first period of the thesis was focused more on theoretical study of the problem. This issue was resolved relatively quickly with the purchase of a suitable computer, which allowed me to continue working independently.

I would like once again to sincerely thank the aforementioned individuals for their understanding and support, as they never caused me any problems and were always considerate.

All of the above are mentioned in order to describe the circumstances under which this thesis was carried out and the challenges I faced in completing it.

Κεφάλαιο 0

Εκτεταμένη Περίληψη

0.1 Εισαγωγή

Η τεχνητή νοημοσύνη είναι ένα από τα πιο καθοριστικά χαρακτηριστικά της εποχής μας, καθώς υπάρχει σε όλους τους τομείς της καθημερινότητάς μας, από την ψυχαγωγία μας, με βίντεο που δημιουργούνται μέσω αυτής μέχρι και σε απλές οικιακές εφαρμογές, όπως τα αυτόματα συστήματα ποτίσματος. Αυτή την τάση της αυτοματοποίησης και χρήσης Τεχνητής Νοημοσύνης ακολουθεί και ο τομέας της οδήγησης, όπου ο συνδυασμός τους, είναι αρκετά ωφέλιμος για όλους τους οδηγούς, διότι αυξάνει την ασφάλεια των οδηγών και καθιστά την εμπειρία οδήγησης πιο ευχάριστη.

Στην οδήγηση, ο απόλυτος στόχος είναι ο έλεγχος του οχήματος εξ ολοκλήρου από τους on-board υπολογιστές αυτού, χωρίς την παρέμβαση του ανθρώπου-οδηγού, δηλαδή η πλήρης αυτονομία. Σε αυτό το σενάριο, ο ρόλος του ανθρώπου αλλάζει από οδηγό με πλήρη έλεγχο σε απλό επιβάτη. Όπως είναι φυσικό, μια τέτοια αλλαγή δεν πραγματοποιείται στιγμιαία, αλλά σταδιακά. Αυτή η πορεία έχει αποτυπωθεί από τη Society of Automotive Engineers (SAE), η οποία είναι μια παγκόσμια κοινότητα και οργανισμός προτύπων, σε έξι στάδια:

- Στο επίπεδο 0, δεν υπάρχει κανένας αυτοματισμός και ο οδηγός είναι υπεύθυνος για τον έλεγχο του οχήματος.
- Στο επίπεδο 1, υπάρχει μια υποβοήθηση του, που σημαίνει ότι το όχημα μπορεί να βοηθήσει τον οδηγό να διατηρήσει σταθερή μια ορισμένη ταχύτητα ή στο να διατηρήσει το όχημα εντός της λωρίδας του.
- Στο επίπεδο 2, μερική αυτονομία έχει επιτευχθεί, όπου το όχημα μπορεί να ελέγχει τη ταχύτητά του και τη θέση του στις λωρίδες, συνήθως σε οδήγηση σε αυτοκινητοδρόμους.
- Στο επίπεδο 3, υπάρχει αυτονομία υπό συνθήκες, όπου το όχημα αποκτά πλήρη έλεγχο, αλλά σε συγκεκριμένα περιβάλλοντα και είναι υπεύθυνο αυτό για την επιτήρηση του περιβάλλοντο του. Ωστόσο, μπορεί οποιαδήποτε στιγμή να ζητήσει από τον οδηγό να πάρει τον έλεγχο.
- Στο επίπεδο 4, υπάρχει υψηλή αυτονομία σε συγκεκριμένες συνθήκες. Όταν οι συνθήκες αυτές καλύπτονται τότε, το όχημα λειτουργεί με πλήρη αυτονομία χωρίς να υπάρχει ανάγκη για οδηγό.

- Στο επίπεδο 5, έχει επιτευχθεί πλήρης αυτονομία χωρίς περιορισμούς, και ο άνθρωπος έχει μετατραπεί σε επιβάτης.

Τη στιγμή της συγγραφής αυτής της διπλωματικής, το υψηλότερο επίπεδο αυτονομίας που είναι εμπορικώς διαθέσιμο στο κοινό είναι το επίπεδο 3 από την Mercedes-Benz με το σύστημα Drive Pilot, το οποίο είναι περιορισμένο σε κάποιες περιοχές της Γερμανίας με όριο ταχύτητα τα 95 km/h, και σε κάποιες πολιτείες της Αμερικής. Ωστόσο, υπάρχει και το Waymo στις Ηνωμενές Πολιτείες που προσφέρει υπηρεσίες, όπως Robotaxi, φορτηγά και διανομές με οχήματα με αυτονομία επιπέδου 4.

0.1.1 Σκοπός της διπλωματικής

Για να επιτευχθεί η πλήρης αυτονομία, το όχημα πρέπει να μπορεί να αντιλαμβάνεται με ακρίβεια περιβάλλον του και να προβλέπει τις κινήσεις των γύρω οχημάτων.

Σε ένα ουτοπικό σενάριο, που απέχει αρκετά από τα σημερινά δεδομένα, τα οχήματα θα είναι όλα αυτόνομα με V2V (Vehicle-to-Vehicle) επικοινωνία, όπου κάθε όχημα θα επικοινωνεί με τα κοντινά του και θα μοιράζονται πληροφορίες σχετικά με τις προθέσεις του, και δεν θα υπάρχει κίνδυνος. Στην πραγματικότητα όμως, όπου τα περισσότερα οχήματα ελέγχονται από ανθρώπους, μπορούν να γίνουν πολύ εύκολα λάθη που να οδηγήσουν σε ατύχημα και ένα από αυτό είναι οι αλλαγές λωρίδας χωρίς τη χρήση των φωτεινών ενδείξεων που είναι εξοπλισμένα τα οχήματα.

Σε αυτή τη διπλωματική εστιάζουμε στην πρόληψη τέτοιων ατυχημάτων, προβλέποντας τέτοιες μανούβρες με βάση το πως κινούνται τα γειτονικά οχήματα στο χώρο. Δεδομένου ότι δεν υπάρχει μια διαδεδομένη, εμπορικά διαθέσιμη λύση του προβλήματος αυτού, το καθιστά ένα ενδιαφέρον θέμα για έρευνα.

Οι περισσότερες υπάρχουσες προσεγγίσεις αυτού του προβλήματος βασίζονται στη θέση του οχήματος στο χώρο ή σε δεδομένα που δεν μπορούν να αποκτηθούν από άλλο όχημα όπως η εφαρμογή του γκαζιού και του φρένου, η θέση του τιμονιού και το οπτικό πεδίο του οδηγού. Η εργασία επιχειρεί να αντιμετωπίσει το πρόβλημα αυτό βασιζόμενη στις δύο συνιστώσες της ταχύτητας, δηλαδή στη διαμήκη και στην πλευρική ταχύτητα. Χρησιμοποιώντας αυτά τα 2 χαρακτηριστικά, θα εκπαιδεύσουμε ένα μοντέλο μηχανικής μάθησης με στόχο την πρόβλεψη επικείμενων αλλαγών λωρίδας.

Για την εκπαίδευση ενός μοντέλου μηχανικής μάθησης απαιτείται ένας τεράστιος όγκος δεδομένων ώστε να καλύπτονται όσο το δυνατόν περισσότερα σενάρια γίνεται. Αυτή η ανάγκη για πολλά δεδομένα έχει οδηγήσει εταιρείες όπως η Tesla να χρησιμοποιούν μεγάλης κλίμακας συλλογή δεδομένων από τον στόλο οχημάτων τους ώστε να βελτιώσουν τους αλγορίθμους αυτόνομης οδήγησης. Σύμφωνα με τη Tesla, το Full Self Driving σύστημά της είναι εκπαιδευμένο σε δισεκατομμύρια μίλια ανώνυμων πραγματικών δεδομένων οδήγησης που αντιστοιχούν σε πάνω από 100 χρόνια σεναρίων οδήγησης και έχουν ανακτηθεί από τον στόλο τους που είναι πάνω από 6 εκατομμύρια οχήματα.

Σε αυτή τη διπλωματική, την ανάγκη για τεράστιο όγκο δεδομένων την αντιμετωπίζουμε κάνοντας χρήση συνθετικών δεδομένων που παρήχθησαν από τον προσομοιωτή CARLA, ενώ ακολούθησε αξιολόγηση του μοντέλου σε dataset πραγματικών δεδομένων: NGSIM US Highway 101 και German Autobahn Dataset.

Οι συνεισφορές μας είναι οι ακόλουθες:

- Προτείνουμε έναν νέο τρόπο πρόβλεψης για επικείμενες μανούβρες, χρησιμοποιώντας Διαμήκεις και Πλευρικές ταχύτητες και Μηχανική Μάθηση
- Δείχνουμε ότι μοντέλα που έχουν εκπαιδευτεί με συνθετικά δεδομένα που έχουν παραχθεί μέσω εκτεταμένων προσομοιώσεων, μπορούν να πετύχουν υψηλό επίπεδο ακρίβειας σε πραγματικά δεδομένα
- Παρουσιάζουμε μια νέα ολοκληρωμένη διαδικασία προεπεξεργασίας για δεδομένα από τον προσομοιωτή CARLA, η οποία αυτοματοποιεί τον μετασχηματισμό των raw δεδομένων του προσομοιωτή σε δομημένες χρονοσειρές κατάλληλες για είσοδο μοντέλων με σκοπό την εκπαίδευση και την αξιολόγησή τους.

0.2 Θεωρητικό Υπόβαθρο

0.2.1 Τεχνητό Νευρωνικό Δίκτυο

Τα Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ) είναι ένα υπολογιστικό σύστημα επεξεργασίας δεδομένων, το οποίο προσομοιώνει την λειτουργία του ανθρώπινου εγκεφάλου. Αυτά τα συστήματα, αποτελούνται από διασυνδεδεμένους κόμβους που ονομάζονται τεχνητοί νευρώνες, που αντιστοιχούν στους βιολογικούς νευρώνες του εγκεφάλου και είναι τοπολογικά οργανωμένοι σε επίπεδα.

Οι νευρώνες συνδέονται μεταξύ τους με ακμές, που με τη σειρά τους αντιστοιχούν στις συνάψεις του εγκεφάλου. Όπως και στον εγκέφαλο, όπου ένας νευρώνας εκπέμπει ένα σήμα σε έναν άλλον μέσω της σύναψής τους, έτσι και στα ΤΝΔ, κάθε νευρώνας λαμβάνει ένα σύνολο αριθμητικών εισόδων από άλλους νευρώνες, τις μετασχηματίζει βάσει ενός γραμμικού συνδυασμού με ανάλογα βάρη και ύστερα από την εφαρμογή μιας μη-γραμμικής συνάρτησης ενεργοποίησης παράγει την τελική έξοδο, η οποία τροφοδοτείται στην συνέχεια σε άλλους νευρώνες του δικτύου.

Το βασικό στοιχείο των ΤΝΔ είναι η δυνατότητα εκπαίδευσής τους, μέσω μιας διαδικασίας που ονομάζεται Μηχανική Μάθηση, η οποία σταδιακά βελτιώνει την ικανότητά τους να επιλύουν το πρόβλημα για το οποίο δημιουργήθηκαν.

0.2.2 Ανατροφοδοτούμενα Νευρωνικά Δίκτυα

Τα Ανατροφοδοτούμενα Νευρωνικά Δίκτυα (ΑΝΔ - Reccurent Neural Networks (RNNs)) είναι ΤΝΔ ειδικά σχεδιασμένα για την επεξεργασία ακολουθιών δεδομένων, όπως κείμενο, ομιλία και χρονοσειρές, όπου η σειρά και η χρονική δομή των στοιχείων είναι σημαντικά. Σε αντίθεση με δίκτυα όπου η πληροφορία κινείται προς μια κατεύθυνση, τα ΑΝΔ χρησιμοποιούν συνδέσμους ανατροφοδοσίας, όπου η έξοδος μιας χρονικής στιγμής ενός νευρώνα, ανατροφοδοτείται ως είσοδος στο δίκτυο σε επόμενη στιγμή. Αυτό επιτρέπει στο ΑΝΔ να αναγνωρίζει χρονικές εξαρτήσεις και μοτίβα μέσα σε ακολουθίες.

Λόγω του σχεδιασμού τους, τα ΑΝΔ αντιμετωπίζουν σημαντικές προκλήσεις όταν προκύπτουν μακροπρόθεσμες εξαρτήσεις. Κατά τη διάρκεια της εκπαίδευσης, οι τιμές παραγώγων

του σφάλματος μπορεί είτε να μηδενίζονται είτε να παίρνουν πολύ μεγάλες τιμές οδηγώντας στο πρόβλημα εξαφάνισης ή έκρηξης των παραγώγων (Vanishing/Exploding Gradient problem).

0.2.3 Δίκτυα Μακράς Βραχύχρονης Μνήμης

Τα Δίκτυα Μακράς Βραχύχρονης Μνήμης (ΔΜΒΜ - Long Short-Term Memory (LSTM)) είναι ειδική μορφή ΑΝΔ, που σχεδιάστηκαν με σκοπό να αντιμετωπίσουν τον περιορισμό που είχαν αυτά στο να μάθουν μακροπρόθεσμες εξαρτήσεις εξαιτίας του προβλήματος εξαφάνισης ή έκρηξης των παραγώγων. Τα ΔΜΒΜ έχουν μια πιο σύνθετη εσωτερική δομή που τους επιτρέπει να διατηρούν και να διαχειρίζονται καλύτερα την πληροφορία σε μεγαλύτερες ακολουθίες.

Ο πυρήνας ενός κόμβου ΔΜΒΜ αποτελείται από ένα κελί μνήμης που είναι υπεύθυνο για την αποθήκευση πληροφορίας σε αυθαίρετα χρονικά διαστήματα. Η ροή της εισερχόμενης και εξερχόμενης πληροφορίας ελέγχεται από τρεις πύλες. Την πύλη εισόδου, την πύλη εξόδου, και την πύλη διαγραφής. Η πύλη εισόδου ελέγχει πόση καινούρια πληροφορία θα αποθηκεύτεί στο κελί μνήμης, η πύλη διαγραφής ελέγχει πόση πληροφορία από τη προηγούμενη κατάσταση θα διαγράψει, ενώ η πύλη εξόδου ελέγχει το ποσοστό της πληροφορίας του κελιού που θα δωθεί ως έξοδος.

0.2.4 Συνθετικά Δεδομένα

Συνθετικά Δεδομένα αποκαλούμε τα δεδομένα που έχουν παραχθεί με τεχνητό τρόπο έτσι ώστε να προσομοιώνουν χαρακτηριστικά και μοτίβα που εμφανίζονται σε αντίστοιχα πραγματικά δεδομένα. Τα τελευταία χρόνια, τα συνθετικά δεδομένα έχουν γίνει ένα σημαντικό κεφάλαιο στον τομέα της μηχανικής μάθησης, ειδικά στις περιπτώσεις όπου τα πραγματικά δεδομένα είναι δύσκολο ή ακριβό να αποκτηθούν.

Ένα βασικό προτέρημά τους είναι η ελαστικότητα που προσφέρουν, αφού επιτρέπουν στους ερευνητές να προσομοιώσουν επικίνδυνες ή σπάνιες καταστάσεις και να αποκτήσουν δεδομένα που σε διαφορετική περίπτωση θα έθεταν την σωματική τους ακεραιότητα σε κίνδυνο ή θα έχαναν πολύτιμο χρόνο. Επίσης, είναι χρήσιμα σε περιπτώσεις όπου τα πραγματικά δεδομένα παρουσιάζουν ανισορροπία μεταξύ κλάσεων, διότι δίνεται η δυνατότητα παραγωγής δεδομένων για τις κλάσεις με τα λιγότερα δείγματα. Ωστόσο, έχουν και μειονεκτήματα, όπως η πιθανή αδυναμία σύλληψης της πολυπλοκότητας και του θορύβου που υπάρχουν στον πραγματικό κόσμο.

Στην παρούσα διπλωματική εργασία, η χρήση συνθετικών δεδομένων επιτρέπει την αντιμετώπιση των δυσκολιών συλλογής πραγματικών δεδομένων, όπως ακριβοί αισθητήρες, πολύς χρόνος και σε ορισμένες περιπτώσεις αυξημένη επικινδυνότητα.

0.2.5 Προσομοιωτής CARLA

Ο τρόπος που επιλέχθηκε για την παραγωγή των συνθετικών δεδομένων είναι ο προσομοιωτής CARLA (Car Learning to Act), ο οποίος είναι ένας προσομοιωτής οδήγησης ανοιχτού κώδικα (open-source), που υλοποιήθηκε με σκοπό την υποστήριξη της διαδικασίας εκπαίδευσης, αξιολόγησης και μέτρησης της απόδοσης αυτόνομων συστημάτων οδήγησης. Ο

βασικός του στόχος είναι να κάνει την έρευνα γύρων από την αυτόνομη οδήγηση προσβάσιμη σε μεγαλύτερο κοινό, χωρίς την ανάγκη μεγάλων επενδύσεων, προσφέροντας ένα ισχυρό, ευέλικτο και ρεαλιστικό ψηφιακό περιβάλλον.

Η επιλογή του έγινε καθώς, είναι το μόνο ενεργό project προσομοιωτή που πληρούσε τις προϋποθέσεις της παρούσας έρευνας. Αυτές, περιελάμβαναν φωτορεαλιστικά γραφικά, που πιθανώς να χρειαστούν σε μελλοντικές επεκτάσεις αυτής της έρευνας, ακριβή φυσική προσομοίωση (physics) που προσφέρουν υψηλό ρεαλισμό, ευκολία χρήσης, καθώς, και μια μεγάλη και ενεργή κοινότητα χρηστών που διευκολύνει την επίλυση τυχών προβλημάτων. Υπήρχαν δύο ακόμα προσομοιωτές που κάλυπταν αυτές τις προϋποθέσεις αλλά έχει σταματήσει η ανάπτυξή τους, αυτοί ήταν ο AirSim και ο LGSVL.

Ένα από τα σημαντικότερα χαρακτηριστικό του προσομοιωτή CARLA είναι το σύστημα Traffic Manager, το οποίο ελέγχει τη συμπεριφορά των οχημάτων που είναι μέρος της προσομοίωσης. Ο Traffic Manager επιτρέπει τη δημιουργία ρεαλιστικών συνθηκών κυκλοφορίας μέσω της ρύθμισης παραμέτρων όπως η πιθανότητα αλλαγής λωρίδας, η ταχύτητα, η υπακοή στα φανάρια και η τυχαιότητα των διαδρομών, χωρίς να απαιτείται ο προγραμματισμός των οχημάτων ένα προς ένα.

Επιπλέον, ο CARLA προσφέρει μια μεγάλη ποικιλία έτοιμων χαρτών πόλεων, ο καθένας με διαφορετικά χαρακτηριστικά, ενώ δίνει και τη δυνατότητα χρήσης άλλων χαρτών που έχουν παραχθεί με χρήση ειδικών προγραμμάτων. Τέλος, προσφέρει τη δυνατότητα τροποποίησης των καιρικών συνθηκών, επιτρέποντας την αξιολόγηση μοντέλων υπό διάφορες και δύσκολες συνθήκες, οι οποίες θα ήταν δύσκολο ή επικίνδυνο να αναπαραχθούν σε πραγματικές συνθήκες οδήγησης.

0.3 Μοντέλο Συστήματος και Περιβάλλον Προσομοίωσης

0.3.1 Σύνολο Δεδομένων

Για την παραγωγή του συνόλου δεδομένων, υλοποιήσαμε μια νέα ολοκληρωμένη διαδικασία προεπεξεργασίας, η οποία χωρίζεται σε 5 στάδια. Το πρώτο στάδιο, είναι αυτό κατά το οποίο ο προσομοιωτής καταγράφει σε ένα Memory-mapped αρχείο διάφορες τιμές σχετικές με το κάθε όχημα της προσομοίωσης, όπως η θέση, η ταχύτητα, η επιτάχυνση και ο προσανατολισμός. Αυτό το αρχείο περιέχει τις τιμές αυτές αποθηκευμένες με τη δεκαεξαδική τους μορφή.

Το δεύτερο στάδιο, είναι ένα custom Python script, το οποίο μεταφράζει την δεκαεξαδική μορφή των τιμών που βρίσκονται στο αρχείο που περιγράψαμε προηγουμένως, και δημιουργεί ένα νέο αρχείο CSV με τα δεδομένα αυτά σε μορφή φιλική προς το χρήστη. Το script αυτό δεν πραγματοποιεί καμία μετατροπή στις ίδιες τις τιμές, παρά μόνο στον τρόπο απεικόνισής τους.

Το τρίτο στάδιο, είναι ακόμα ένα custom Python script, το οποίο επεξεργάζεται την έξοδο του προηγούμενου σταδίου και δημιουργεί ένα νέο αρχείο CSV, το οποίο περιέχει χαρακτηριστικά υψηλότερου επιπέδου, καθώς και μεταφέρει αυτούσια κάποια χαρακτηριστικά χωρίς επεξεργασία. Τα νέα χαρακτηριστικά είναι η πλευρική και η διαμήκη ταχύτητα σύμφωνα με τη σύμβαση ότι η κίνηση προς τα δεξιά έχει θετικό πρόσημο και η κίνηση προς τα αριστερά

αρνητικό.

Στο τέταρτο στάδιο, αλλάξαμε την δομή που ήταν αποθηκευμένα τα δεδομένα. Αρχικά ήταν αποθηκευμένα σε μια διάταξη που δυσκόλευε τη χρήση τους, κατά την οποία μια γραμμή αντιστοιχούσε σε μια χρονική στιγμή και οι στήλες της γραμμής αυτής ήταν όλες οι πληροφορίες όλων των οχημάτων. Εμείς αλλάξαμε τη δομή αυτή, έτσι ώστε, κάθε γραμμή να αντιστοιχεί στις τιμές ενός οχήματος για μια χρονική στιγμή και το κάναμε με τέτοιο τρόπο ώστε οι τιμές κάθε οχήματος να είναι συνεχόμενες και, όταν τελειώνουν οι γραμμές του ενός, τότε να αρχίζουν του επόμενου. Επίσης, όταν τελείωναν οι γραμμές του ενός οχήματος, πριν προσθέσουμε τις γραμμές ενός άλλου, προσθέταμε ενδιάμεσα 10 γραμμές στις οποίες όλες οι στήλες είχαν τιμή -10, με σκοπό να σηματοδοτήσουμε την αλλαγή.

Στη συνέχεια, επεξεργαστήκαμε τη στήλη που είναι αποθηκευμένο το αναγνωριστικό της λωρίδας που κινείται το όχημα. Από την ανάλυση των δεδομένων βρέθηκε ότι όταν το όχημα κινείται προς τα αριστερά, η απόλυτη τιμή του αναγνωριστικού της λωρίδας μειωνόταν κατά ένα, ενώ σε αντίθετη περίπτωση, όταν το όχημα κινείται προς τα δεξιά, η απόλυτη τιμή του αναγνωριστικού της λωρίδας αυξανόταν κατά ένα. Με βάση αυτή τη παρατήρηση προχωρήσαμε στον υπολογισμό της διαφοράς δύο συνεχόμενων απόλυτων τιμών, και καταλήξαμε με 3 τιμές, -1 όταν το όχημα πηγαίνει προς τα αριστερά, 0 όταν διατηρεί τη λωρίδα και 1 όταν κινείται προς τα δεξιά. Υπήρχαν και στήλες με τιμές διαφορετικές από αυτές τις 3, όπου με περαιτέρω ανάλυση βρέθηκε ότι αντιστοιχούν σε άλλες μανούβρες όπως έξοδο και είσοδο σε αυτοκινητοδρόμους ή στροφές. Σε αυτές τις στήλες αλλάξαμε τη τιμή τους σε -10 με σκοπό να σηματοδοτήσουμε λανθασμένη εγγραφή. Τέλος, η τιμή των στηλών που ήταν -1 αλλάχθηκε σε 2 διότι το framework μηχανικής μάθησης που χρησιμοποιήσαμε, απαίτει ως αναγνωριστικά κλάσεων θετικούς ακέραιους αριθμούς.

Το πέμπτο και τελευταίο στάδιο, είναι αυτό της δημιουργίας κατάλληλων εισόδων για το μοντέλο. Τα χαρακτηριστικά που επιλέχθηκαν θέλαμε να υπολογίζονται εύκολα και με αξιοπιστία από ένα όχημα σε ένα πραγματικό σενάριο. Για τους λόγους αυτούς επιλέξαμε τη πλευρική και διαμήκη ταχύτητα. Το μοντέλο μας χρησιμοποιεί ΔMBM ως στρώμα εισόδου το οποίο χρειάζεται ως είσοδο χρονοσειρά συγκεκριμένου μήκους. Για να το πετύχουμε αυτό θα δημιουργήσουμε υπο-σειρές με τη τεχνική του μετακινούμενου παραθύρου (sliding window), το μήκος του οποίου θα το βρούμε με τη τεχνική Βελτιστοποίησης υπερπαραμέτρων, Grid Search.

Από ανάλυση δεδομένων βρήκαμε ότι ο CARLA καταγράφει μια μανούβρα, όταν το μέσο του οχήματος βρίσκεται ακριβώς πάνω από τη διαχωριστική γραμμή, χρονική στιγμή που είναι αργά για την πρόβλεψη μανούβρας. Για να αντιμετωπίσουμε αυτόν τον περιορισμό δεν χαρακτηρίσαμε ως μανούβρα μόνο τη χρονοσειρά που τελειώνει ακριβώς πριν την καταγραφή του CARLA, αλλά και τις χρονοσειρές που τελείωναν κάποια χρονικά βήματα πριν και μετά από αυτή. Τα ιδανικά αυτά βήματα θα τα βρούμε, πάλι, εφαρμόζοντας την τεχνική του Grid Search.

Τέλος, από τη διαδικασία δημιουργίας παραθύρων αποκλείστηκαν οι εγγραφές που περιείχαν τις τιμές -10 που είχαμε περιγράψει στα προηγούμενα στάδια.

0.3.2 Συλλογή δεδομένων

Η συλλογή δεδομένων έγινε σε δύο διαφορετικούς χάρτες, όπου τα δεδομένα που πήραμε από την προσομοίωση στον πρώτο χάρτη (Town04) χρησιμοποιήθηκαν στη διαδικασία εκπαίδευσης και αξιολόγησης (evaluation), ενώ τα δεδομένα από την προσομοίωση με τον δεύτερο χάρτη (Town10) χρησιμοποιήθηκαν στη διαδικασία τελικής αξιολόγησης (Testing) του μοντέλου μας. Η πρώτη προσομοίωση διήρκησε 4.5 ώρες και περιείχε 41 οχήματα, ενώ η δεύτερη διήρκησε 1.5 ώρα και περιείχε 11 οχήματα. Σε κάθε προσομοιώση χρησιμοποιήθηκε διαφορετικός ¨σπόρος' (seed) για τη τυχαιότητα του Traffic Manager, ενώ κάθε όχημα είχε διαφορετικές και τυχαίες τιμές στις πιθανότητες αλλαγής λωρίδας προς τα δεξιά και αριστερά και διαφορετικό και τυχαίο όριο ταχύτητας. Ο ρυθμός λήψης δεδομένων ορίστηκε στα 5Hζ.

0.3.3 Μοντέλο Μηχανικής Μάθησης

Η αρχιτεκτονική του μοντέλου μας δεν είναι πολύ σύνθετη, αλλά είναι αρκετή για να αποδώσει καλά και σε συνθετικά αλλά και σε πραγματικά δεδομένα. Χρησιμοποιήσαμε, 2 επίπεδα ΔMBM, το ένα μετά το άλλο, με σκοπό να αναγνωρίζει και βραχυπρόθεσμες μεταβολές αλλά και υψηλότερου επιπέδου χρονικά μοτίβα. Και στα δύο αυτά επίπεδα εφαρμόστηκε Dropout, όπου ένα ορισμένο, από τον χρήστη, ποσοστό των κόμβων μηδενίζουν τα βάρη τους κάθε εποχή, που βοηθάει στην αντιμετώπιση της υπερπροσαρμογής στα δεδομένα εκπαίδευσης (overfitting). Στη συνέχεια, ακολουθεί ένα επίπεδο πλήρως διασυνδεδεμένων κόμβων (Fully Connected) με συνάρτηση ενεργοποίησης την LeakyReLU που βοηθάει το μοντέλο να ξεχωρίζει χαρακτηριστικά χωρίς το πρόβλημα του ανενεργού νευρώνα, και τέλος, υπάρχει το τελικό επίπεδο πλήρως διασυνδεδεμένων κόμβων με συνάρτηση ενεργοποίησης την Softmax που είναι υπεύθυνο για την τελική ταξινόμηση της χρονοσειράς εισόδου σε μια από τις τρεις κλάσεις.

0.3.4 Διαδικασία Εκπαίδευσης

Η διαδικασία εκπαίδευσης ήταν αρκετά απλή, όπου χρησιμοποίησαμε ως βελτιστοποιητή τον αλγόριθμο Αδαμ (Adam Optimizer), συνάρτηση απώλειας την Categorical Cross-Entropy και ως μετρική αξιολόγησης ορίσαμε την ακρίβεια (accuracy). Η χρήση της Categorical Cross-Entropy, απαιτεί τις κλάσεις να έχουν την one-hot κωδικοποίηση, για την επίτευξη αυτού χρησιμοποιήσαμε την συνάρτηση *to_categorical* από την βιβλιοθήκη Tensorflow.keras.utils. Ως μέγιστος αριθμός εποχών ορίστηκε το 100, ενώ έγινε χρήση της επιλογής για πρόωρο τερματισμό της εκπαίδευσης (EarlyStopping Callback) αν η τιμή της ακρίβειας στα δεδομένα της αξιολόγησης δεν αυξανόταν τουλάχιστον κατά 0.01 για 10 συνεχόμενες εποχές.

0.4 Μεθοδολογία

0.4.1 Τελική Αξιολόγηση (Testing)

Η τελική αξιολόγηση με δεδομένα που δεν έχει ξαναδεί το μοντέλο μας έγινε με τη χρήση τριών διαφορετικών συνόλων δεδομένων, τα οποία είναι: συνθετικά δεδομένα από διαφορετικό

χάρτη, ένα σύνολο πραγματικών δεδομένων με όνομα US Highway 101 Dataset από το Υπουργείο Μεταφορών των Ηνωμένων Πολιτειών, και ακόμα ένα σύνολο πραγματικών δεδομένων από τον γερμανικό αυτοκινητόδρομο Autobahn, δημοσιευμένο σε μια εργασία ερευνητών του πανεπιστημίου Aachen της Γερμανίας. Τα δύο σύνολα πραγματικών δεδομένω για να χρησιμοποιηθούν χρειάστηκαν μια προεπεξεργασία αντίστοιχη με αυτά των συνθετικών δεδομένων για να τα φέρουμε σε κατάλληλη μορφή.

Για το καθένα από αυτά τα σύνολα δεδομένων θα πραγματοποιηθούν δύο ξεχωριστά σενάρια αξιολόγησης, στο πρώτο θα μελετήσουμε την απόδοση του μοντέλου μας στην πρόβλεψη μανούβρων σε μικρούς ορίζοντες πρόβλεψης που έχουμε μεγάλη απαίτηση για υψηλά ποσοστά ακρίβειας, ενώ στο δεύτερο θα χρησιμοποιήσουμε αρκετούς ορίζοντες πρόβλεψης ξεκινώντας από 2.4 δευτερόλεπτα πριν την καταγραφή της μανούβρας μέχρι και 0.8 μετά για να έχουμε μια πλήρη εικόνα της απόδοσής τους.

Οι μετρικές αξιολόγησης που θα χρησιμοποιήσουμε είναι οι αρκετά διαδεδομένες Accuracy, Recall, Precision, F1-Score.

0.5 Αποτελέσματα

0.5.1 Αποτελέσματα Βελτιστοποίησης υπερπαραμέτρων

Μετά την εκτέλεση όλων των Grid Search βρήκαμε ότι το ιδανικό μήκος παραθύρου είναι τα 5 χρονικά βήματα ή διαφορετικά, παράθυρα μήκους ενός δευτερολέπτου, ενώ για τη δημιουργία του συνόλου δεδομένων, τα καλύτερα αποτελέσματα τα πήραμε όταν χρησιμοποιήσαμε σύνολο δεδομένων που περιείχε 5 χρονοσείρες για την κάθε μανούβρα οι οποίες ήταν:

1. Από $t_{start} = -2.0s$ εώς $t_{end} = -0.8s$ από τη καταγραφή του CARLA
2. Από $t_{start} = -1.8s$ εώς $t_{end} = -0.6s$ από τη καταγραφή του CARLA
3. Από $t_{start} = -1.6s$ εώς $t_{end} = -0.4s$ από τη καταγραφή του CARLA
4. Από $t_{start} = -1.4s$ εώς $t_{end} = -0.2s$ από τη καταγραφή του CARLA
5. Από $t_{start} = -1.2s$ εώς $t_{end} = 0.0s$ από τη καταγραφή του CARLA

0.5.2 Αποτελέσματα Εκπαίδευσης

Η εκπαίδευση διήρκησε 16 εποχές, αντί των 100 που είχαν αρχικά οριστεί, εξαιτίας του μηχανισμού πρόωρης διακοπής. Η καλύτερη απόδοση σημειώθηκε στην 14η εποχή όπου η ακρίβεια στο σύνολο δεδομένων της αξιολόγησης έφτασε το 95.7% ενώ η τιμή της συνάρτησης απώλειας ήταν 0.1537.

0.5.3 Αποτελέσματα Τελικής Αξιολόγησης

Τα αποτελέσματα της τελικής αξιολόγησης ήταν αρκετά ικανοποιητικά στο σύνολο συνθετικών δεδομένων και στο σύνολο πραγματικών δεδομένων από τον γερμανικό Αυτοκινητόδρομο, όπου το μοντέλο μας πέτυχε 92.6% και 86.2% αντίστοιχα στη μετρική συνολικής

ακρίβειας, ενώ στο σύνολο πραγματικών δεδομένων US Highway 101 ήταν ελαφρώς χειρότερα με 69.1% στο πρώτο σενάριο αξιολόγησης.

Για το δεύτερο σενάριο αξιολόγησης η εικόνα δεν άλλαξε πολύ, αφού το μοντέλο μας είχε καλή απόδοση στο σύνολο συνθετικών δεδομένων και σε μεγαλύτερους ορίζοντες πρόβλεψης, φτάνοντας περίπου στο 50% συνολικής ακρίβειας στα 1.6 δευτερόλεπτα πριν τη μανούβρα, στο σύνολο δεδομένων US Highway 101 έφτασε στο 50% συνολικής ακρίβειας στα 2.0 δευτερόλεπτα, ενώ στο σύνολο δεδομένων του γερμανικού αυτοκινητόδρομου δεν έπεσε κάτω από 50% ακόμα και στα 2.4 δευτερόλεπτα πριν τη μανούβρα.

0.5.4 Υπολογιστική Επίδοση

Το προτεινόμενο μοντέλο, εκτός από ικανοποιητική ακρίβεια στις προβλέψεις, παρουσιάζει και υψηλή υπολογιστική επίδοση, όπου ο μέσος χρόνος που χρειάζεται για να κάνει μια πρόβλεψη είναι μόλις 0.15 ms χρησιμοποιώντας μόνο CPU, που αυτό σε ένα αληθινό σενάριο θα μας επιτρέπει να χρησιμοποιήσουμε την GPU για συστήματα με μεγαλύτερη ανάγκη για παραλληλοποίηση.

0.6 Επίλογος

0.6.1 Σύνοψη και Συμπεράσματα

Όπως είδαμε και στην ενότητα των αποτελεσμάτων, καταφέραμε να δημιουργήσουμε ένα μοντέλο εκπαιδευμένο αποκλειστικά με συνθετικά δεδομένα που πετυχαίνει ικανοποιητικά αποτελέσματα στις προβλέψεις με πραγματικά δεδομένα. Ωστόσο, παρατηρούμε ότι σε μεγαλύτερους ορίζοντες πρόβλεψης, η απόδοση του μοντέλου μας φθίνει αρκετά. Αναλύοντας τα δεδομένα που είχαμε στην διάθεσή μας και κάνοντας τη σύγκριση αληθινών και συνθετικών δεδομένων παρατηρούμε ότι τα συνθετικά δεδομένα έχουν λιγότερο θόρυβο από τα αληθινά, όπως άλλωστε ήταν αναμενόμενο. Συγκεκριμένα, μια μανούβρα στα συνθετικά δεδομένα χρειάζεται πολύ μικρό χρονικό διάστημα, από τη στιγμή που θα ξεκινήσει μέχρι και τη στιγμή που θα ολοκληρωθεί, ενώ μάλιστα, πριν ληφθεί η απόφαση αλλαγής λωρίδας, το όχημα μένει σταθερό στο κέντρο της λωρίδας του, σε αντίθεση με τα αληθινά δεδομένα, όπου η μανούβρα παίρνει περισσότερο χρόνο και γίνεται πιο ομαλά.

Συνοπτικά, σε αυτή τη διπλωματική δείξαμε ότι ένα μοντέλο εκπαιδευμένο με συνθετικά δεδομένα, μπορεί να αποδώσει αρκετά καλά και σε πραγματικές συνθήκες, ενώ επιβεβαιώσαμε ότι η δημιουργία αλγορίθμων μηχανικής μάθησης απαιτεί έναν τεράστιο όγκο δεδομένων, αφού σε δύο πραγματικά σενάρια υπό παρόμοιες συνθήκες, με τη μόνη φανερή διαφορά να είναι η χώρα προέλευσης των δεδομένων, το ίδιο μοντέλο είχε μια σημαντική απόκλιση στην απόδοσή του. Συνεπώς, σε αυτόν τον τομέα, η χρήση σωστά ορισμένων προσομοιωτών για παραγωγή δεδομένων μπορεί να αποδειχθεί ένα πολύ καλό εργαλείο.

0.6.2 Δυσκολίες και Περιορισμοί

0.6.3 Μελλοντικές Επεκτάσεις

Η παρούσα εργασία επιδέχεται διάφορες μελλοντικές επεκτάσεις. Η πρώτη είναι η διασύνδεση του μοντέλου μας με τον προσομοιωτή, όπου θα γίνονται προβλέψεις σε πραγματικό χρόνο, και για να γίνει αυτό απαιτείται περαιτέρω μελέτη του CARLA για την κατάλληλη ενσωμάτωση των Python scripts για τη δημιουργία των είσοδων του μοντέλου.

Μια άλλη επέκταση με μεγαλύτερο ενδιαφέρον, είναι η τροποποίηση του Traffic Manager με σκοπό την χρήση διαφορετικών παραμέτρων των PID ελεγκτών στο κάθε όχημα, έτσι ώστε να έχουμε ακόμα μεγαλύτερη ποικιλία στις συμπεριφορές.

Chapter 1

Introduction

Artificial intelligence and the automation of many human activities through it are the defining characteristics of our era [8]. From the simplest daily task, such as watering one's flowers, to more complex and demanding ones, such as the construction of objects and even houses, processes have been automated and, in combination with artificial intelligence, have been optimized. For example, watering flowers was initially performed automatically without any specific criteria; today, irrigation systems take into account data such as soil moisture, soil type, and the particular needs of each plant.

From this broader trend of automation and autonomy in human activities, driving could not be excluded. Driving may be considered a hobby and an enjoyable experience by some, but for others it is a boring activity they would rather avoid, while for others it is a profession. Full or partial autonomy could be highly beneficial for all drivers, as it would both enhance safety by minimizing human error and make the driving experience more enjoyable.

Already in the 1970s and 1980s, the global automotive industry had begun investing in optimizing vehicle performance through the introduction of various sensors, such as Lambda sensors, to increase efficiency and reduce emissions [9]. At the same time, significant efforts were made to enhance driver safety through the installation of airbags and collision sensors [10].

Subsequently, Advanced Driver Assistance Systems (ADAS) were launched, initially with simple sensors, such as the Parktronic[11] system, which assists drivers in parking. Over time, these systems evolved to include functionalities such as maintaining a constant speed without driver input, alerting the driver in cases of unintentional lane departure, and gradually achieving ever higher levels of automation.

This continuous investment is also reflected in the market value of automotive sensors, which was estimated at $2.3 billion in 1991 and $7.0 billion in 2006 [12], while in 2024 it was projected to have reached $27.4 billion [13].

The goal of this automation is, of course, complete vehicle control by the onboard computer system without human intervention. In this case, the human stops being the driver and instead becomes a passenger. Naturally, this transition does not occur instantly but is structured into levels of automation. There are six levels defined by the Society of Automotive Engineers (SAE) International [1] as shown in Figure 1.1, a global professional association and standards organization, which are outlined briefly below.

- At Level 0, no automation exists, and the driver is fully responsible for controlling the vehicle.
- At Level 1, driver assistance is introduced, meaning the vehicle can support the driver by maintaining a constant speed (cruise control) or by assisting in keeping the vehicle within a lane.
- At Level 2, partial automation is achieved: the vehicle can control both speed and lateral positioning within the lane simultaneously, typically in highway driving scenarios.
- At Level 3, conditional automation is enabled: the vehicle assumes full control in certain environments and, crucially, becomes responsible for monitoring the driving environment itself. However, it may request at any moment that the driver take back control.
- At Level 4, high automation is present under specific conditions. When these conditions are satisfied, the vehicle operates with complete autonomy; otherwise, control reverts to the driver, or the vehicle remains stationary. Such conditions are usually tied to geographic constraints, speed limits, or weather. At this stage, vehicles may even lack a steering wheel or pedals.
- Finally, at Level 5, full automation exists without any restrictions. The human is exclusively a passenger, and vehicles could be designed without steering wheels or pedals.

At the time of writing this thesis, the highest level of autonomous driving available to the public in commercially sold vehicles is Level 3, achieved by Mercedes-Benz through its *Drive Pilot* system [14]. However, this system is currently restricted to certain regions in Germany and operates at a maximum speed of 95 km/h, with plans to extend this limit to 130 km/h by 2030. The system has also been approved in the states of California and Nevada of the United States of America.

However, the most widely publicized company in the field of autonomous driving is Tesla, with its Full Self-Driving (FSD) system, which, despite its name, is classified only as Level 2. Finally, in the United States, Waymo, formerly known as the Google Self-Driving Car Project, offers Robotaxi, Trucking, and Delivery services with vehicles that are categorized at Level 4, representing the highest level of autonomous driving achieved so far.

1.1 Scope of this Thesis

One of the main challenges in achieving the highest possible level of autonomy lies in the environment in which the vehicle operates. This environment is dynamic with constant changes, with the most significant and unpredictable factor being the surrounding vehicles, especially when controlled by humans, because as shown in a study from the U.S

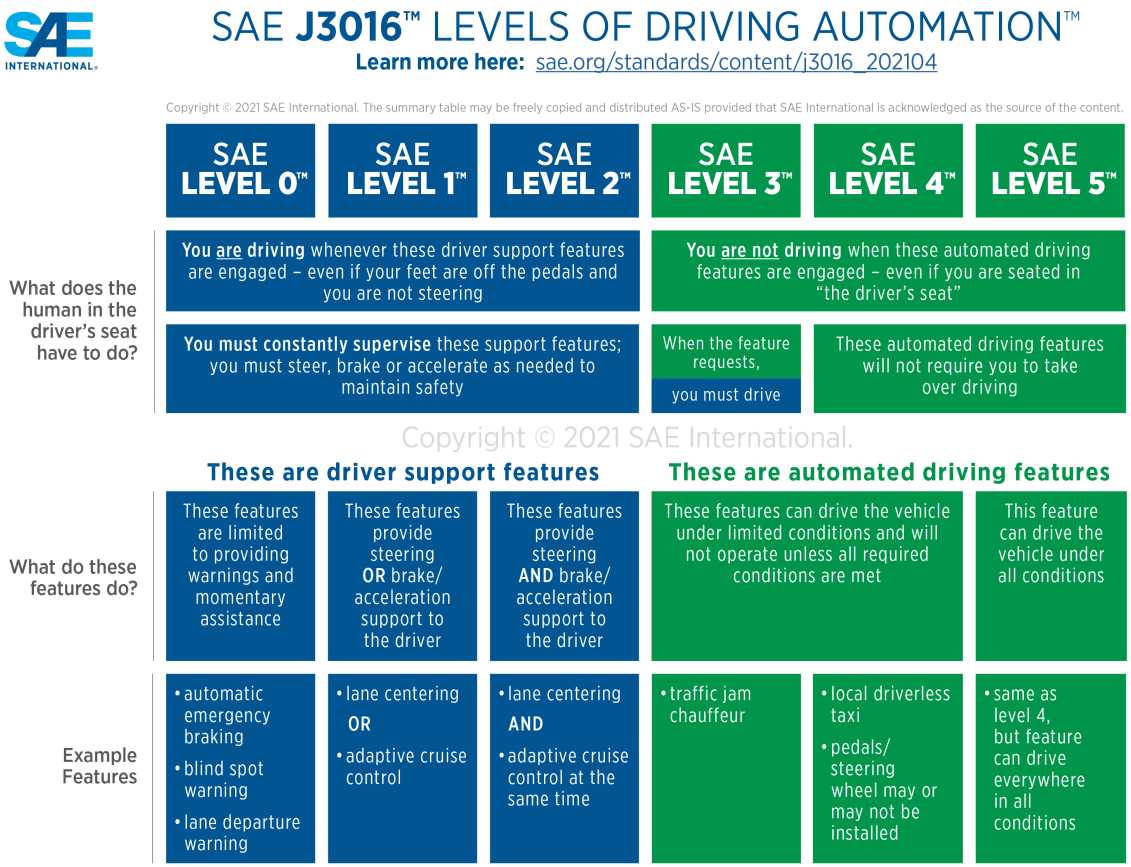

Figure 1.1. *SAE International Levels of Driving Automation [1]*

Department of Transportation, the critical reason for 94% of motor vehicle crashes is human error[15]. When there are no nearby vehicles, the situation is relatively simple: the autonomous car only needs to follow the traffic rules and the designated route. However, once other vehicles enter the equation—especially when they are controlled by human drivers—the difficulty increases dramatically.

In a utopian scenario, far from today's reality, all vehicles would be autonomous and equipped with Vehicle-to-Vehicle (V2V) communication-where each vehicle shares it's data like speed, direction and intentions with other vehicle close to it. In such a system, each vehicle would continuously inform its neighbors about its intentions, allowing all of them to cooperate seamlessly and achieve complete safety with no risk of accidents. Clearly, this scenario is still a long way ahead. At present, V2V communication does not exist in practice, and the majority of vehicles on the roads are driven by humans, who are highly prone to errors. Some of the most common mistakes leading to accidents include failing to pay attention to the surroundings through the mirrors, changing lanes without using indicators, driving under the influence of alcohol, and falling asleep at the wheel due to fatigue.

This thesis focuses on preventing accidents caused by lane changes that occur without the use of turn signals. To avoid such incidents, an upcoming maneuver must be predicted as early as possible, based solely on how nearby vehicles move in space. Since there is

no widespread, commercially available solution to this problem, it remains an interesting topic for research.

Existing approaches are based either on the absolute position of a vehicle in space or on data that cannot be accessed by other vehicles, such as throttle and brake input, steering wheel angle, or the driver's field of view. In this work, the problem is addressed using only the two components of speed—longitudinal and lateral velocity. Using these two features, a machine learning model is trained to predict imminent vehicle maneuvers.

Existing approaches are based either on the absolute position of a vehicle in space or on data that cannot be accessed by other vehicles, such as throttle and brake input, steering wheel angle, or the driver's field of view. In this work, the problem is addressed using only the two components of speed—longitudinal and lateral velocity. These quantities directly describe the vehicle's motion in the direction of travel and across lanes, making them reliable indicators of maneuver intent. Moreover, they can be easily obtained from standard sensors, which makes this approach more practical and independent of vehicle-specific control data. Using these two features, a machine learning model is trained to predict imminent vehicle maneuvers.

Training a machine learning model requires a large and diverse dataset in order to capture as many potential scenarios as possible. This need for large datasets has led major industry players, such as Tesla, to rely on massive data collection from their vehicle fleets to improve autonomous driving algorithms. According to Tesla, its Full Self-Driving system is trained on billions of miles of anonymous real-world driving data, which is over one hundred years of driving scenarios, collected from a fleet of more than six million vehicles[16]. In this thesis, this challenge is addressed by the use of synthetic data generated by the CARLA simulator, followed by an evaluation of the trained model on real-world driving datasets NGSIM US Highway 101[17] and German Autobahn Dataset[18].

In summary, this thesis aims to address the prediction of upcoming maneuvers by using a different set of vehicle features than existing approaches and by employing synthetic data instead of relying on small-sized real-world datasets that lack a consistent structure.

Our contribution is the following:

- We propose a novel way of predicting upcoming maneuvers by utilizing Longitudinal and Lateral Velocities in a Machine Learning Model.
- We demonstrate that models trained with synthetic data generated through extensive simulations can achieve high level of accuracy in real-world data.
- We introduce a novel end-to-end preprocessing pipeline for CARLA simulator data, that automates the transformation of raw simulator output into structured time-series inputs for model training and evaluation.

1.2 Structure of this Thesis

This thesis is organized into six chapters. Chapter 1 introduces the objectives of autonomous driving, describes the main pillars of a typical autonomous system, presents the

levels of autonomy, and briefly outlines the goal of this work. Chapter 2 reviews related research efforts on the development of vehicle behavior prediction systems, while Chapter 3 provides the theoretical background necessary for understanding and describing the problem, the simulation environment, and the programming mechanisms used to implement the machine learning algorithm. Chapter 4, System Model and Simulation Environment, describes the dataset pipeline, the process of collecting training and testing data, and the design and training of the proposed model. Chapter 5, Methodology, outlines the experimental scenarios, the preprocessing of the real-world dataset, and the evaluation metrics used in the study. Chapter 6 presents the obtained results, while Chapter 7, Epilogue, discusses the conclusions, limitations, and possible directions for future work.

Part I

Theoretical Part

Chapter 2

Literature Review

The problem of vehicle behavior prediction has been extensively studied in recent years and can be approached from multiple perspectives, depending on the available computational resources, the nature and quality of the data, and the technological framework adopted. Each formulation of the problem implies a distinct methodological approach.

2.1 Predicting Trajectory

One common formulation involves predicting the precise future position of a vehicle over a defined time horizon. This formulation is considered the most demanding, as it requires high precision in a problem characterized by numerous external factors that are not solely related to the state of the vehicle itself. Such factors include the current traffic conditions, the driving behavior of individual drivers, as well as the topology of the road.

Several reasons led to the exclusion of this formulation from the present research. First, this type of prediction is highly sensitive to noise and measurement errors, where even a minor sensor inaccuracy can negatively influence the predicted path. Second, the accumulation of uncertainty and error can cause the prediction to deviate significantly when forecasting several steps into the future, which explains why studies consistently report a decline in performance as the prediction horizon increases. Third, the precise future position is inherently dependent on the road topology, further increasing the complexity of any model that must account for it. Finally, in decision-making systems, maneuver prediction provides sufficient semantic information without the computational cost associated with estimating exact coordinates.

The authors of [19] propose a simple architecture consisting of an LSTM layer with 256 memory cells, followed by two TimeDistributed fully connected layers with 256 and 128 neurons, respectively, and a final fully connected output layer with 2 neurons corresponding to the position x_{targ} and velocity u_{targ}. Furthermore, the initial input of the network was also passed as an argument to the final output layer. For training purposes, the authors used the US101 dataset from NGSIM [17]. They utilized 4892 trajectories for training and reserved the remaining 1209 for testing. The initial architecture described above yielded promising results, with the RMS (Root-Mean-Square) error in position reaching $0.4m$ for a prediction horizon of 4 seconds and $0.73m$ for a 10-second horizon, while the corresponding RMS error in velocity was 1.49 m/s at 4 seconds and 2.96 m/s at 10 seconds.

The authors of [20] proposed a different approach, in terms of both model architecture and input representation, where the model is not limited to information from a single vehicle but instead incorporates knowledge of the surrounding area. Specifically, they employ an encoder-decoder architecture, which they enhance through the use of a Convolutional LSTM (ConvLSTM). This variant of LSTM replaces the traditional element-wise (Hadamard) product with convolution operations. The input of the decoder is the concatenation of the outputs of the encoder and the ConvLSTM. The encoder receives as input an Occupancy Map, a matrix of weights representing how much a vehicle located at position (i, y) of the map influences the behavior of the ego vehicle. In parallel, ConvLSTM receives a Risk Map, which serves as an auxiliary input to generate the context vector for the decoder. This Risk Map encodes the likelihood of the ego vehicle occupying the same position as another vehicle, thus modeling potential collision risks. The proposed network consists of an encoder with 128 units and a batch size of 32, the other input layer consist by two stacked ConvLSTM layers with 128 hidden states each. The decoder includes 128 LSTM cells, followed by a fully connected layer. The authors evaluate their model on two separate datasets: the US101 dataset from NGSIM [17] and the HighD dataset [21]. On the former, they report RMSE values ranging from 0.41 to 3.87 for prediction horizons of 1 to 5 seconds, respectively, while on the latter, the RMSE ranges from 0.22 to 2.8 over the same prediction intervals.

Another approach is presented in [22], where the authors propose a graph-based spatio-temporal convolutional network (GSTCN) designed to predict the future trajectory distributions of all vehicles within a scene simultaneously. The scene is defined as the ego vehicle's horizon of sight, which includes the two adjacent lanes and extends ± 100 meters longitudinally. The elements of this grid are inversely proportional to the distance between any two vehicles belonging to the same scene. To capture spatial dependencies, a graph convolutional network (GCN) is used [23], while temporal dependencies are captured using a CNN-based temporal dependency extractor (TDE). Finally, a GRU-based encoder-decoder network is used to generate the future trajectory distributions. Training and evaluation are conducted on the US-101 [17] and I-80 [24] datasets from NGSIM, achieving RMSE values ranging from 0.44 to 2.98 for prediction horizons between 1 and 5 seconds.

2.2 Predicting Maneuver

An alternative formulation of the problem focuses on predicting the maneuver that a vehicle will perform within a given time horizon. This formulation is comparatively simpler and less computationally demanding, but also less precise, as it does not aim to determine the exact location of the vehicle, but rather a broader area in which it will be positioned. The most common maneuver classes include lane change to the right, lane change to the left, lane keeping, right turn, and left turn.

The authors of [25] propose a rather sophisticated approach, which, in addition to data related to the vehicle's dynamic and kinematic state, also incorporates information regarding the driver's cephalo-ocular behavior. Specifically, they employ two features: the

horizontal head movement of the driver and the three-dimensional Point of Gaze (PoG), which is obtained using a binocular eye-gaze tracker. For their experiments, they collected and used their own dataset. The model architecture consists of three LSTM layers, each with 100 units, followed by a fully connected output layer with 5 units corresponding to the maneuver classes, using Softmax as the activation function. They also trained an additional model with the same architecture, except that the final fully connected layer contained only 3 units, representing the same classes but excluding the two turning maneuvers, thereby retaining only lane changes and lane keeping. According to their results, the models achieved 82% and 76% accuracy for the 3-class and 5-class settings, respectively, over a prediction horizon of 3.5 seconds.

The researchers of the previous work built upon the approach introduced in [26], where, instead of employing machine learning techniques, a statistical model was used, specifically Hidden Markov Models. As in the later study, they relied on their own dataset. The reported performance achieved 79.5% precision and 83.3% recall over a prediction horizon of 3.8 seconds.

Another study [27] adopts a hybrid system, in which the maneuver is first recognized and the vehicle's trajectory is subsequently estimated. Unlike the previously mentioned works that focused on five common maneuvers, the researchers in this study identified ten. The maneuver types considered include: crossing from an adjacent lane, overtaking—where the following vehicle changes lane and passes the ego vehicle—cut-ins, where a vehicle from a neighboring lane overtakes and moves into the ego lane ahead of it, and finally, drift into the ego's lane from the left or the right, where a vehicle enters the ego lane behind it. The ten maneuvers arise from considering the two possible directions for each of the five types.

For training, they used a custom dataset they created themselves by equipping a vehicle with various sensors and recording 45 minutes of driving. They report two sets of results: one that disregards the topology of surrounding vehicles, and another that incorporates it. In the first case, they achieved a correct classification rate of 83.49%, while in the second, the rate increased slightly to 84.24%. The reported runtime is 0.0891 s for the first configuration and 0.1546 s for the second; however, this refers to the entire pipeline, including the trajectory prediction stage.

In [28], the authors follow a conceptually similar hybrid framework to that of the aforementioned study, in which the maneuver is first identified and the trajectory is subsequently estimated. However, their approach differs fundamentally in methodology. First, they model the vehicle's path as an arc of a circle, and then they represent the three relevant lanes—left, current, and right. They proceed by computing the statistical distances between the modeled vehicle path and each lane, and based on these distances, they infer the maneuver the vehicle is likely to perform. This heuristic method achieves a 100% detection rate 1.1 seconds before the actual lane change. Nevertheless, the authors do not explicitly define the precise moment at which a lane change is considered to have occurred; it could correspond either to the instant the vehicle is exactly over the lane marking or the point at which it has fully crossed it.

In summary, most existing solutions rely on expensive sensors to generate their own datasets. As a result, these approaches are evaluated on data that are not publicly available,

which prevents direct comparison with other methods. In contrast, our solution generates its own data using a free simulator, allowing the data collection process to be reproduced without costly equipment. Furthermore, some solutions do not use machine learning, but rely instead on statistical or heuristic approaches, which are less capable than LSTM-based algorithms for handling data with temporal dependencies. Another limitation of existing solutions is the lack of reporting on the time required to predict such maneuvers, which is an important factor for practical deployment. Finally, our method uses different features for prediction, specifically vehicle dynamics, providing a better representation of vehicle behavior.

Chapter 3

Theoretical Background

3.1 Timeseries

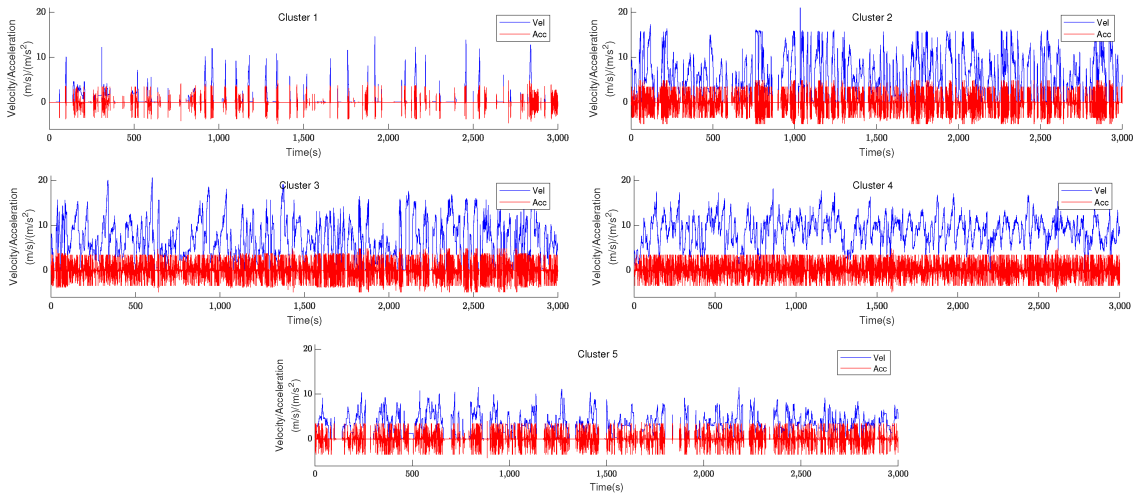

Figure 3.1. *Time-series visualization of vehicle velocity and acceleration from the Next Generation Simulation (NGSIM) database which contain data collected from four US highways. Image from [2].*

3.1.1 Definition of Time Series Data

Time-series is a sequence of data points listed in time order where the data points are recorded in consistent intervals, making them discrete-time data. Time-series data have an inherent temporal dependency, which means that future values are influenced by past values. Time series are used in many areas like statistics, signal processing and in many domains of engineering. Using time-series analysis we can extract useful statistics and other characteristics of the data.

3.1.2 Characteristics of Time Series Data

Time-series data have several key characteristics that differentiate them from other types of data, and also appear on the data we will be using, some of them are temporal dependency, seasonality, autocorrelation, and structural breaks. Having natural temporal ordering makes their analysis different from cross-sectional studies, in which there is no natural ordering, for example, explaining people's wages by reference to their respective education levels. Seasonality makes the data regular with repeating patterns that occur at fixed intervals. Autocorrelation is the correlation of a time-series with its past values which means that past observations can be used to predict future values. Finally, Structural breaks are the abrupt changes in the data pattern due to external factors.

3.1.3 Common Time Series Models

As mentioned before, past observations of time-series influence future values, by taking advantage of this characteristic various models have been developed to analyze and make

predictions on such data, ranging from statistical approaches to deep learning methods. One of the most fundamental models is the Auto-regressive (AR) model which is most useful when the future values are linearly dependent on past values. However, most times this is not the case and future values also have some random fluctuations or errors and the Moving Average (MA) models become better. The Moving Average models break-down the current value as a function of past error terms and by doing that it captures and smooths out noise in the data.

By combining both models and also differencing to achieve stationarity (I) we get the Autoregressive Integrated Moving Average (ARIMA) model[29]. This model is effective for univariate time series displaying linear dependencies and non-stationarity, making it widely applicable in econometrics and other domains.

However, real-world time series data often display complex, non-linear variatons. This has led to the broad use of more flexible models like Long Short-Term Memory (LSTM) networks which is a type of Recurrent Neural Network (RNN) designed to capture, as its name suggests, long-term dependencies in time series data. LSTMs are particularly advantageous in scenarios where time series exhibit intricate temporal dependencies, such as in financial market forecasting [30] or natural language processing applications[31]. LSTM will be the main focus for the remainder of this study.

Ultimately, the choice of an appropriate time series model depends on several factors, including the presence of seasonality, the length of historical data, and the degree of non-linearity in the time series. While classical statistical models remain relevant for many applications, advances in deep learning continue to push the boundaries of what is possible in time series forecasting.

3.2 Machine Learning

Machine learning (ML) is a subset of artificial intelligence (AI) that focuses on developing algorithms capable of learning patterns from data and making decisions on unseen data without being explicitly programmed. The algorithm's ability to change its parameters based on data is where "Learning" comes from. Unlike traditional rule-based systems, ML models improve their performance over time by learning from examples, making them highly suitable for complex tasks such as vehicle behavior prediction.

3.2.1 Types of Machine Learning

Machine Learning can be categorized into 3 main categories:

- Supervised Learning algorithms learn a function that can be used to predict the output associated with a given input. The algorithm is being trained on a set of inputs and the desired output
- Unsupervised Learning algorithms identify commonalities in the given data and labels, classifies or categorizes it. The algorithm is trained on inputs without labels, so the algorithm can find patterns on its own.
- Reinforcement Learning, where the algorithm learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties.

Supervised Learning

As mentioned above, in Supervised Learning the model is trained on a set of examples where each example is a pair of input-output. Usually the input is the raw or processed data while the output is the transformation of the input which the model has to find. The output varies based on the task the algorithm has to perform. For example, if we want the algorithm to perform Image Classification then the input will be an array of pixels and the output will be the category in which the image belongs.

Depending on the output, supervised learning can be divided in classification and regression algorithms. Classification algorithms are used when the outputs are restricted to a limited set of values, while regression algorithms are used when the outputs can take any numerical value within a range. For example, the Image Classification task is a classification task while predicting a person's height based on factors like genetics is a regression task.

Unsupervised Learning

Unsupervised Learning algorithms find similarities in data that has not been labeled, classified or categorized. There algorithms make decisions based on the absence or presence of certain commonalities in each new piece of data. Most common applications of unsupervised algorithms are clustering, dimensionality reduction and density estimation. In clustering the algorithm

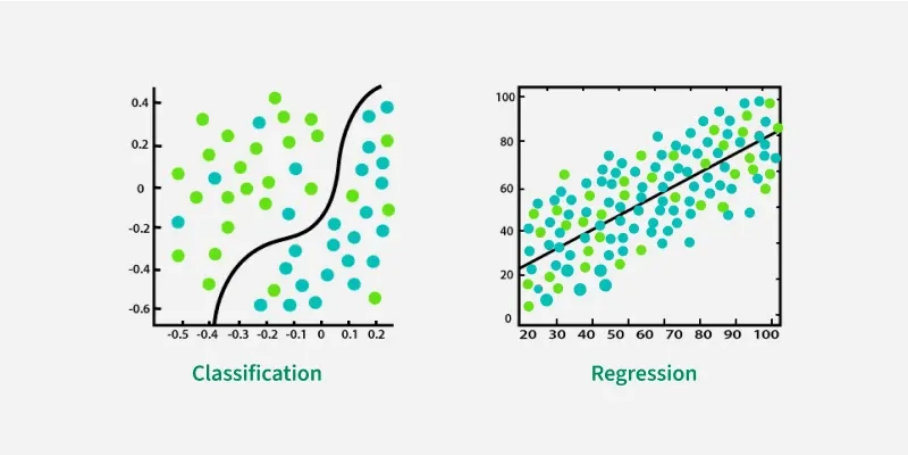

Figure 3.2. *Visual comparison between classification and regression. The left panel shows a classification task with a decision boundary separating two classes (e.g., disease vs. healthy). The right panel shows a regression task with a best-fit line predicting a continuous outcome (e.g., survival years). Image from [3].*

3.2.2 Key Concepts

Dataset

In machine learning, a dataset is a structured collection of data used to develop, train, and evaluate predictive models. Each entry in the dataset is typically composed of two main components: features and labels. Features, also known as input variables, represent the measurable attributes or characteristics of the system being modeled. These inputs are used by the learning algorithm to infer patterns or relationships within the data. Labels, also known as target variables, represent the desired outputs that the model aims to predict based on the given features.

To ensure proper training and evaluation of the model, datasets are commonly divided into two or three separate subsets. The training set is used to fit the model by feeding it labeled examples, allowing it to learn the underlying structure of the data by adjusting its parameters. After each training step the model is fed with the validation set to get an overview of the training progress on unseen data, to guide hyperparameter tuning and also make decisions like Early Stopping to avoid overfitting. Lastly, the test set is reserved for evaluating the performance of the final model in previously unseen examples, providing insight into its ability to generalize. The overall quality, size, and balance of the dataset are critical factors that directly affect the performance and reliability of the resulting model.

Feature Selection and Engineering

Feature selection and feature engineering are two separate and fundamental processes in the preparation of data for machine learning models. Their goal is to improve model performance by identifying the most relevant information from the input data and transforming it into a form that best supports learning.

Feature selection refers to the process of choosing a subset of the available input variables that contribute meaningfully to the predictive task. By eliminating redundant, irrel-

evant, or noisy features, this process can improve model interpretability, reduce overfitting, and decrease computational complexity. In high-dimensional datasets, such as those encountered in simulation environments or sensor-based data collection, feature selection is particularly important for isolating the most informative signals.

Feature engineering, on the other hand, involves the creation or transformation of features to better represent the underlying problem. Raw data collected from real-world or simulated environments may not be immediately suitable for learning algorithms. For instance, rather than using only the raw values of variables such as as speed or direction, derived quantities such as their rates of change or relative differences between agents can offer richer contextual information. These transformations often require domain-specific knowledge and creativity, as they aim to expose patterns that are not directly visible in the original inputs.

In combination, feature selection and engineering form a bridge between raw data and meaningful representations, enabling models to learn more efficiently and make more accurate predictions. These steps are especially crucial when dealing with time-series or spatial-temporal data, where dynamic relationships often carry more importance than static values.

Data Preprocessing

Data preprocessing is one of the most important steps in data mining and model training processes. Data collection, especially from a simulation, is subject to out-of-range values, impossible data combinations, and missing values. Also raw data sometimes may be useless for training a machine learning algorithm, for example, many times the rate of change of a variable is much more valuable than the value of the variable itself. Preprocessing is the process by which unstructured or useless data are transformed into useful and intelligible representations suitable for machine learning algorithms.

In machine learning, a **model** refers to a mathematical representation of a system that is trained to make predictions or decisions based on input data. The model learns patterns and relationships from the data during the training process and uses this knowledge to generalize to new unseen data.

Hyperparameter Tuning

Hyperparameters are configuration variables that do not affect prediction ability like internal parameters, like weights and biases do. Hypeparameters are used to manage the training process and are manually set before the start. Some well known examples are the number of nodes and layers, learning rate, batch size and epochs.

Hyperparameter tuning is a process in which a different set of hyperparameters is passed to the model to be trained with them and because every combination is tested this process is computationally intense and time-consuming. There are two commonly used methods to perform this tuning. The first one is Grid Search where a list of hyperparameters and a performance metric is set and the algorithm traverses through all possible combinations

and determines the best one based on the metric. The second one is Random Search which is not as thorough where only a subset of those combination are tested in random.

Validation, Testing and Inference

Validation and Testing are essential steps in the development and evaluation of a machine learning model, while inference takes place once the final model is deployed on a production environment. They are fundamental for assessing the model's prediction performance, detecting overfitting and guiding hyperparameter tuning. These two processes look the same but they are not. They both feed the model with unseen data and evaluate the model's performance during that stage. The difference is the stage at which each process is performed.

Validation is performed during the training process and it used to help tune the model. It is used for Hyperparameter Tuning and Model Selection, also, many times the loss of the validation is used to determine if the model generalizes better in each epoch or not and if not the training is stopped to avoid overfitting.

Testing, on the other hand, is performed after the model is trained and no more tuning is done. The test set remains unseen during training and validation and provides an objective measure of the model's generalization ability. The metrics that are extracted during testing, such as accuracy, precision, recall, or F1 score, offer insight into how well the model will perform in real-world data.

Inference, in contrast, refers to the stage after testing, when the trained model is used to make predictions on real-world or production data. No further tuning or evaluation is performed at this point, as the model simply generates outputs based on the learned parameters.

Overfitting

Overfitting is a very common challenge in machine learning, where a model learns patterns specific to the training data rather than capturing the underlying structure of the problem. While an overfitted model has very low value on the loss function on the training set, its performance isn't as good when feeding new, unseen data which means poor generalization.

This phenomenon is usually found when a model is too complex relative to the variability of the training data. Higher complexity models have large number of parameters or layers which in turn allows the model to memorize noise or irrelevant differentiations in the training data that do not represent generalizable features. In those cases, the model begins to look for minor changes in the input and fails to perform in unseen data.

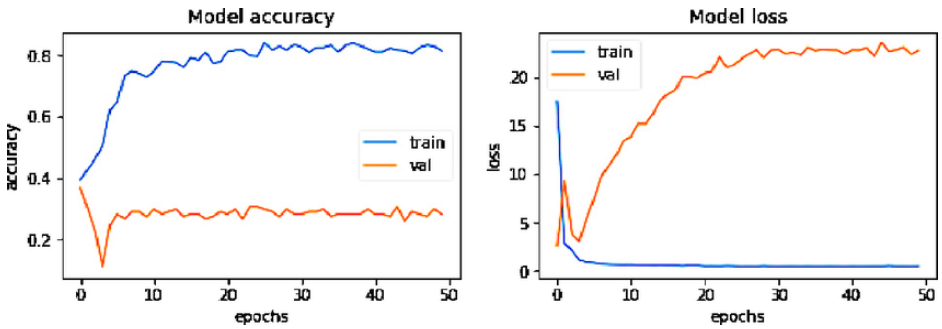

Figure 3.3. *An example of an overfitted model's accuracy and loss curve over epochs of training. As clearly shown, the accuracy on the training set increases while the validation accuracy fluctuates around a fixed value. The opposite applies to the model's loss. Image from [4].*

3.3 Artificial Neural Networks

In machine learning, an artificial neural network, also called a neural network, abbreviated ANN or NN, is a computational model inspired by the structure and functions of biological neural networks. These models consist of connected nodes called artificial neurons, which loosely resemble the biological neurons in the brain, and are organized in multiple layers.

Neurons are connected to each other with edges, which in turn resemble the synapses in the brain. Like the brain, where a neuron transmits a signal to another with the synapse, in an ANN a neuron receives a signal, processes it, and then sends it to its connected neurons. The signal is the output of the neuron computed by its activation function, and the strength of the signal is determined by its weight.

Typically, neurons are grouped into layers and different layers may perform different transformations on their inputs. A signal enters the ANN from the first layer, the input layer, and travels to the last layer, the output layer, most likely passing through multiple intermediate layers, or hidden layers, each layer performing a transformations resulting to the prediction of the model. A network is typically called a Deep Neural Network if it has at least two hidden layers.

3.3.1 Key Concepts

Activation Function

Activation Functions are a fundamental component of ANNs and are responsible for adding non-linearity into a neuron's output. Without them, an ANN composed only of linear transformations would be equivalent to a single-layer linear model, no matter its depth. Non-linearity is essential for enabling the network to approximate complex, real-world functions and to learn intricate patterns within the data.

Each neuron in an ANN computed a weighted sum of all its inputs and then applies an activation function to this result to produce the output of the neuron, or the next layer's input, or the model's final prediction in the case of the output layer. The choice of activation function significantly influences the learning dynamics and overall performance of the network.

The selection of an appropriate activation function depends on the network architecture, the type of task and empirical performance. Understanding their behavior is crucial, especially in deep learning applications where multiple layers interact in complex ways. One of the most widely used functions is the Rectified Linear Unit (ReLU), defined as the positive part of its input. It introduces sparsity and accelerates convergence during training, although it may suffer from the "dying ReLU" problem where neurons can become inactive. Variants such as Leaky ReLU or Parametric ReLU address this limitation by allowing a small, non-zero gradient when the input is negative. Some other commonly used functions are Sigmoid and Softmax, both for tasks such as classification. The first maps the input values to the interval between 0 and 1, thus making it useful for binary classification, while the second outputs a probability distribution over multiple classes and

is typically used in the output layer of multiclass classification networks.

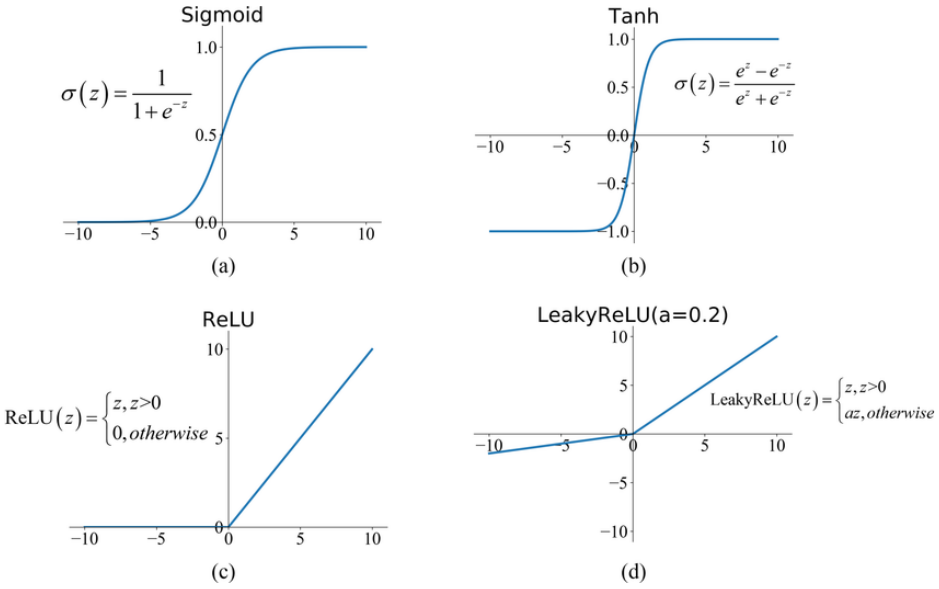

Figure 3.4. *Plot of some of the most common activation functions. Image from [5].*

Training and Parameter optimization

Model Training is the process by which a machine learning algorithm learns to perform a specific task. It is the step in machine learning where the learning occurs. In this step, the model is given input-output pairs from the training dataset and it adjusts its parameters in order to produce more accurate predictions. These parameters, which in ANNs are known as weights and biases, define the behavior of the mathematical representation of the model.

Particularly in ANNs, to correctly adjust these parameters the training process uses relies on a loss function which quantifies the difference between the model's prediction and the true labels. The goal is to minimize this loss function in order to achieve the optimal performance. This minimization is carried out by the use of different optimization algorithms, with the most common choices being Stochastic Gradient Descent (SGD)[32] and Adam[33]. This workflow is repeated until a stopping criterion is met, such as achieving satisfactory performance on a validation dataset or reaching a predefined number of iterations.

Loss Function

The loss function plays a critical role in the predictive model training process. It is used during this process to measure model performance by quantifying the difference between the model's predictions from the correct, "ground truth" labels. By minimizing this loss, the model learns to generalize and make better predictions on unseen data. The choice of loss function depends on the problem we are called to solve.

In classification tasks, which involve assigning input data to discrete categories, commonly used loss functions include categorical cross-entropy and binary cross-entropy. These functions measure the difference between the predicted probability distribution over classes

and the true distribution represented by the ground-truth labels. For regression tasks, where the objective is to predict continuous values, loss functions such as mean squared error (MSE) or mean absolute error (MAE) are often employed.

Ultimately, the effectiveness of an ANN heavily relies on the suitability of the chosen loss function for the specific task. An appropriately selected loss function not only reflects the learning objective accurately but also ensures stable and efficient convergence of the training algorithm

Early Stopping

Early Stopping is a form of regularization used to avoid overfitting. Regularization refers to the process of modifying a learning algorithm and in this case to prevent overfitting. During the training process, the Optimizer, updates the model to make it better fit the training data with each iteration. This improves the performance of the model up to the point of overfitting. Overfitting is usually identified when the value of Loss Function decreases over a set number of epochs but on the other hand the loss of validation stays the same or increases. At this point, training is stopped to conserve computational resources, time and preserve model generalization.

Dropout

Dropout is another form of regularization used to avoid overfitting by randomly deactivating a subset of neurons during each training iteration. Before the start of training a probability is set, commonly between 0.2 and 0.5, that determines how many neurons are going to be temporarily removed from the network during an iteration. These neurons are excluded from forward and backward propagation, which means that the neuron neither contributes to the prediction nor gets its weights updated. However, during testing or deployment the full network is used and the dropout is ignored.

Dropout is particularly effective in deep learning architectures, including Recurrent models such as Long Short-Term Memory (LSTM) networks and although it adds stochasticity to the training process, it is a simple yet powerful method to improve model robustness and is widely adopted in both research and practical machine learning applications.

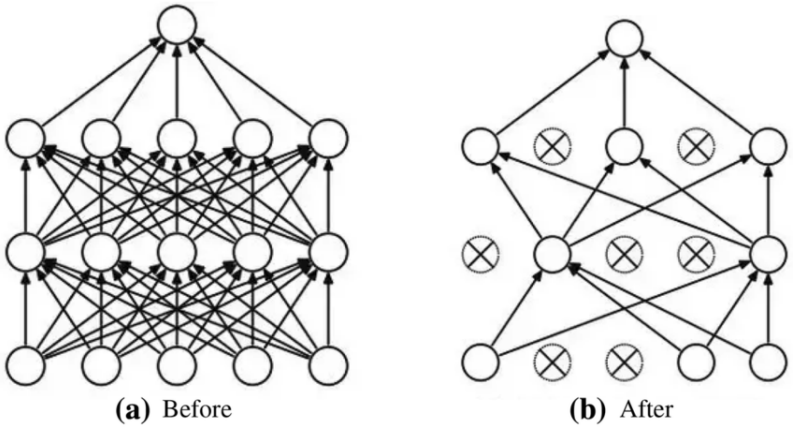

Figure 3.5. *Dropout Application. Image from [6].*

3.3.2 Perceptron

The perceptron, introduced by Frank Rosenblatt in 1958[34], is one of the earliest and simplest models of an artificial neuron and represents the foundation of ANNs. It is a linear binary classifier that takes multiple inputs, multiplies each by a weight, sums the results along with a bias term, and applies an activation function to produce a binary output.

Mathematically, the output of the perceptron y can be described as:

$$y = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i x_i + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

where x_i are the input values, w_i are the weights, and b is the bias term. The activation function in the original perceptron is a simple step function, which outputs either 0 or 1 based on whether the weighted sum is greater than zero.

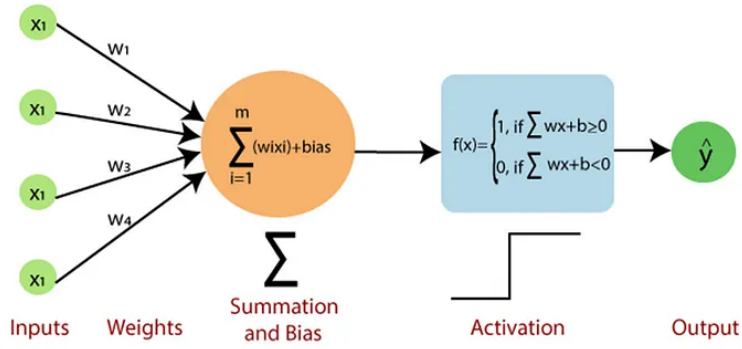

Figure 3.6. *Perceptron. Image from [7].*

Despite its simplicity, the perceptron laid the groundwork for more complex neural architectures. However, it is limited to solving problems that are linearly separable. One well-known example that illustrates this limitation is the inability of a single-layer perceptron to solve the XOR problem. These limitations eventually led to the development of multilayer neural networks, which introduced hidden layers and non-linear activation functions and enabled neural networks to model complex, non-linear relationships.

3.3.3 Multilayer Perceptrons (MLPs)

Multilayer Perceptrons (MLPs) are a class of feedforward neural networks that extend the basic perceptron by introducing one or more hidden layers between the input and output layers. Each layer is composed of multiple artificial neurons, and each neuron is typically connected to every neuron in the next layer, and that is the reason many also call them Fully Connected Networks.

The inclusion of hidden layers and the use of non-linear activation functions, such as the sigmoid or ReLU, allow MLPs to approximate complex, non-linear functions. This capability makes them suitable for a wide range of classification and regression tasks where linear models fall short.

Unlike the single-layer perceptron, MLPs can solve non-linearly separable problems, such as the XOR problem, by transforming the input space into higher-dimensional representations. This transformation enables the network to find decision boundaries that are not limited to straight lines or planes.

Multilayer Perceptrons represent an important milestone in the evolution of neural networks. They form the basis for many modern architectures, even though they do not inherently handle sequential data or temporal dependencies, which are essential in certain domains such as speech recognition, natural language processing, and vehicle behavior prediction.

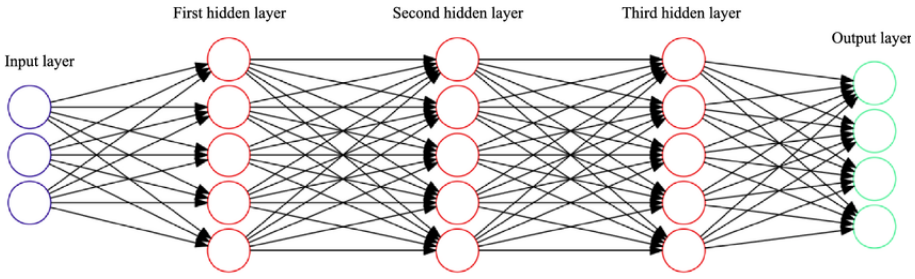

Figure 3.7. *A Multilayer Perceptron or Fully Connected Neural Network. Image from [6].*

3.3.4 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are Artificial neural networks specifically designed for processing sequential data, such as text, speech, and time series where the order and temporal structure of elements is important. Unlike feedforward networks such as Multilayer Perceptrons, which process inputs independently, RNNs utilize recurrent connections, where the output of a neuron at one time step is fed back as input to the network at the next time step. This enables RNNs to capture temporal dependencies and patterns within sequences.

The fundamental building block of RNNs is the recurrent unit which maintains a hidden state. This hidden state acts as a form of memory, enabling the network to retain information about past inputs. Formally, the hidden state h_t at time t is updated as follows:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{3.2}$$

where x_t is the input at time t, W_{xh} and W_{hh} are weight matrices, b_h is a bias term, and σ is a non-linear activation functions such as tanh. The output of the RNN at each time step depends not only on the current input but also on the hidden state from the previous time step and can be calculated by:

$$y_t = \phi(W_{hy}h_t + b_y) \tag{3.3}$$

where h_t is the hidden state at time t, W_{hy} is the weight matrix connecting the hidden state to the output, b_y is the output bias term, and ϕ is an activation function appropriate to the task.

This recurrent structure enables RNNs to learn patterns that evolve over time, making them applicable to time series forecasting, speech recognition, language modeling, and behavior prediction. However, standard RNNs face significant challenges when dealing with long-term dependencies. During training, gradients can either vanish or explode, known as the Vanishing Gradient problem, as they are propagated backward through many time steps. This makes it difficult for the network to learn relationships between distant elements in a sequence.

These limitations led to the development of more advanced recurrent architectures, such as the Long Short-Term Memory (LSTM) network, which introduces mechanisms to better preserve and regulate information across longer sequences.

3.3.5 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a specialized type of Recurrent Neural Network (RNN) designed to address the limitations of standard RNNs, particularly their difficulty in learning long-term dependencies due to the vanishing and exploding gradient problems. Introduced by Hochreiter and Schmidhuber in 1997 [35], LSTMs incorporate a more complex internal structure that enables them to better preserve and manage information across longer sequences. The name is made in analogy with long-term memory and short-term memory and their relationship, studied by cognitive psychologists since the early 20th century.

At the core of the LSTM is a memory cell, which is responsible for storing information over arbitrary time intervals. The flow of information into and out of this cell is regulated by three gates: an input gate, an output gate, and a forget gate. Forget gates determine how much of the information from the previous state should be discarded by mapping the previous state and the current input to a value between 0 and 1. A (rounded) value of 1 signifies the retention of the information, and a value of 0 represents discarding. Input gates control how much new information is stored in the current cell state, using the same system as forget gates. Lastly, output gates control how much of the cell state is exposed as the output at the current time step, by assigning a value from 0 to 1 to the information, considering the previous and current states. The activation functions of the gates that maps values from 0 to 1 is usually the sigmoid function.

The operations of the gates of an LSTM unit at time step t can be described by the following equations[35]:

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) && (forget\ gate)\\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) && (input\ gate)\\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) && (output\ gate)
\end{aligned}
$$

While the cell input candidate $\tilde{c}_t$, the cell state update c_t and the output h_t can be calculated as:

$$\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$h_t = o_t \odot tanh(c_t)$$

where x_t is the input at time t, h_{t-1} is the previous hidden state, c_{t-1} is the previous cell state, $\odot$ denotes element-wise multiplication, σ is the sigmoid activation function, $tanh$ is the hyperbolic tangent function and W_*, U_* and b_* are the learned parameters of the model.

Selectively outputting relevant information from the current state allows the LSTM network to maintain useful, long-term dependencies to make predictions, both in current and future time-steps.

3.4 Synthetic Data

Synthetic data refers to artificially generated data that simulates the characteristics and patterns of real-world datasets. In recent years, it has become an important resource in the field of machine learning, particularly in situations where real data is difficult to obtain, expensive to collect, or sensitive in nature. In this work, the training and evaluation of the predictive model rely on synthetic data generated using the CARLA simulator, a high-fidelity open-source platform designed for autonomous driving research. The simulator allows for the creation of diverse and realistic urban traffic scenarios, enabling the collection of large quantities of labeled data under controlled conditions.

One of the main advantages of synthetic data is its flexibility. It allows researchers to simulate specific situations that are rare or hazardous in the real world, such as building fires [36], wildfires [37] or working with heavy machinery [38], as well as driving scenarios like sudden lane changes or near-collision events, which are the focus of this study. This is particularly relevant for tasks such as maneuver prediction, where rare behaviors like abrupt lane changes are critical to model performance but are underrepresented in real-world datasets. In the case of this study, synthetic data includes vehicle kinematics such as longitudinal speed, and lateral speed, which serve as input features for the model used to predict lane-change maneuvers.

In addition to supporting model training, synthetic data is widely used to address class imbalance by generating samples for underrepresented categories[39] [40] [41]. It also plays a key role in testing and validating machine learning systems, allowing for the evaluation of model behavior under specific conditions without requiring additional data collection efforts. Furthermore, in domains where privacy and data protection are concerns, synthetic data provides a means of sharing useful information while preserving anonymity and compliance with regulations[42] [43] [44]. For example, it can be used to generate artificial datasets that mirror the statistical properties of real data without exposing any sensitive details.

While synthetic data offers many benefits, it also comes with limitations. It may not perfectly capture the complexity, unpredictability, and noise present in real-world environments [45] and may exhibit different statistical and geometric properties than those of real data. This difference is especially noticeable in visual applications, where the quality of camera imagery is affected by these gaps. As a result, models trained solely on synthetic data may experience a performance drop when deployed in real settings. However, when carefully designed and validated, synthetic data can significantly reduce the cost and risk of experimentation without risking real-world generalization [46], especially in fields like autonomous driving, where real-world testing can be dangerous or impractical.

In summary, synthetic data provides a valuable foundation for training, testing, and validating machine learning models. Its use in this thesis enables the efficient development of a predictive model for vehicle behavior in an urban environment while ensuring safety, scalability, and reproducibility of the data collection process.

3.5 CARLA Simulator

The method chosen to obtain synthetic data for this study is the CARLA Simulator[47]. CARLA (Car Learning to Act) is an open-source driving simulator developed to support the training, validation, and benchmarking of autonomous driving systems. Its main goal is to make autonomous driving research more accessible to the wider community by providing a powerful, flexible, and realistic virtual environment. Built on the Unreal Engine, CARLA offers high-quality rendering and physical realism, allowing the simulation of urban and rural environments with detailed maps, dynamic weather conditions, and realistic lighting.

The simulator chosen for this project needed to offer realistic graphics, allowing for future extensions that might require photorealistic rendering, as well as accurate physics to ensure a high level of realism. Finally, it was important to have a large and active community, so that any technical issues could be resolved easily. As we mentioned before, CARLA meets the first two requirements by using one of the most popular and freely available game engines, Unreal Engine 4, which is also used for generating synthetic data[48]. Moreover, the transition to the newer and more advanced Unreal Engine 5 is already underway.

Apart from CARLA, two other simulators also met these criteria: AirSim [49], backed by Microsoft, which is built on the same engine, and LGSVL [50], backed by LG, which uses another widely adopted game engine, Unity. AirSim even has a larger GitHub community with 17,000 stars, compared to CARLA's 13,000. However, neither of these two simulators was selected because both projects have been discontinued by their developers[51].

Figure 3.8 shows the graphical interface of the CARLA simulator used during data collection. The larger window provides a top-down, free-roaming spectator view of the entire highway environment, allowing navigation across the map to monitor all surrounding vehicles. This view can be optionally disabled during long simulation runs in order to reduce rendering load and lower power consumption. The smaller window displays the third-person camera view of the ego vehicle, which may be either manually controlled or operated in autopilot mode. Each road section is annotated with a unique identifier, visualized as red labels on the road surface.

CARLA uses the ASAM OpenDRIVE standard to define road networks and infrastructure, ensuring compatibility with other tools and simulators used in the automotive industry. One of its strengths lies in its robust API support in both Python and C++, which enables researchers to create custom scenarios, control multiple vehicles, access sensor data in real-time, and manage the simulation loop programmatically. The platform is also open to community contributions and is continuously being expanded by an active group of developers and researchers.

A key advantage of CARLA is its ability to simulate a wide variety of sensors commonly used in autonomous vehicles. These include LiDAR, RGB cameras, depth cameras, semantic segmentation cameras, GPS, IMU, radar, and ultrasonic sensors. Each sensor can be configured in terms of position, orientation, frequency, and resolution, offering fine-grained control over the data collection process. Multiple sensors can be attached to a single vehicle, or to multiple vehicles at once, allowing the generation of rich, multi-perspective datasets. To illustrate this flexibility, two screenshots are shown in Figure 3.9, which were

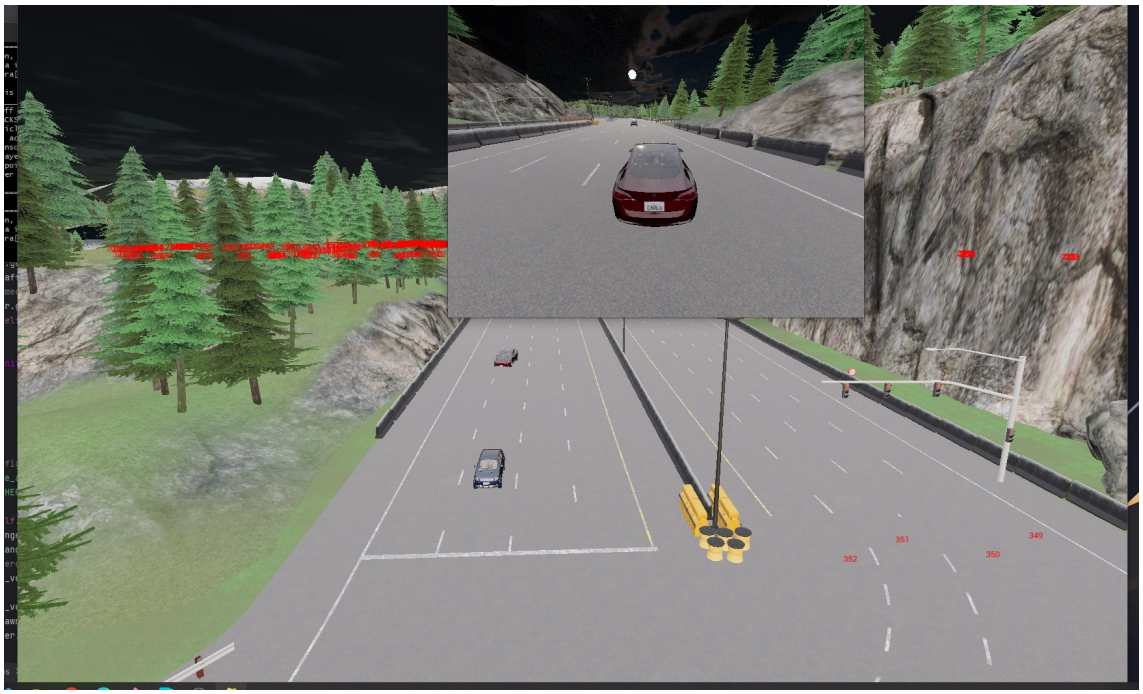

Figure 3.8. *Screenshot from one of our CARLA simulation runs showing a Lane Change.*

captured from the same simulation setup in which only the ego vehicle's camera type was switched, from an RGB camera to a semantic segmentation camera, while the environment, vehicles and spectator view remained unchanged. In both screenshots, a radar sensor was also enabled, with its detections visualized as small white points around nearby vehicles. In addition, CARLA provides access to low-level vehicle information such as position, velocity, acceleration, yaw, and control signals (throttle, brake, steering), making it suitable for both perception and control research.

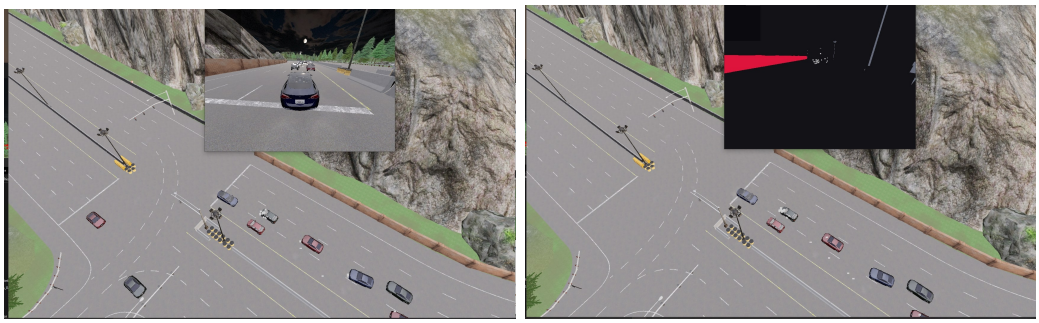

Figure 3.9. *Screenshot from one of our CARLA simulation runs showing different sensors view.*

Another important feature is the Traffic Manager system, which governs the behavior of non-player vehicles (NPCs). The Traffic Manager allows users to simulate realistic traffic flow by adjusting parameters such as speed, lane-changing behavior, distance-keeping, traffic light compliance, and randomization of routes. This enables the simulation of complex, dynamic driving environments without the need for manual scripting of each vehicle. Scenarios can be further customized using the Scenario Runner module, which allows the

definition of specific driving tasks and test cases for benchmarking autonomous agents under reproducible conditions.

CARLA also supports a wide range of predefined maps and the integration of custom-built maps created using external tools like RoadRunner or Blender. Environmental conditions such as rain, fog, cloudiness, time of day, and sun angle can be manipulated dynamically, allowing for the testing of models under varied and challenging scenarios that would be difficult or dangerous to reproduce in real life.

Overall, CARLA provides a comprehensive and safe platform for generating the diverse datasets required to train and evaluate machine learning models for autonomous driving. One notable example is the KITTI-CARLA dataset [52], where researchers created a virtual twin of the real-world KITTI dataset [53] using identical sensors and placement configurations. Another study [54] generated a dataset featuring objects that are underrepresented in real-world data, such as rare or uncommon street scenarios. In a further work [55], a dedicated dataset was developed for evaluating SLAM algorithms. Together, these examples demonstrate the remarkable versatility of CARLA.

In this work, it enables the collection of time-series data related to vehicle dynamics, such as yaw, longitudinal speed, and lateral speed, across multiple scenarios. The simulator's flexibility and sensor fidelity make it an ideal choice for research into vehicle behavior prediction in complex urban environments.

Part II

Experimental Part

Chapter 4

System Model and Simulation Environment

4.1 Dataset

The dataset was generated using a custom pipeline, presented schematically in Figure 4.1 and described in detail in the following sections.

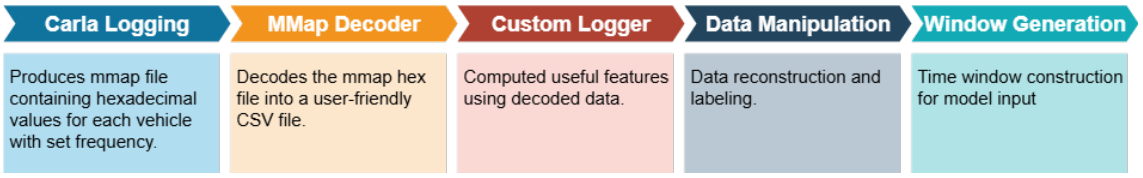

Figure 4.1. *Custom pipeline for dataset generation*

4.1.1 Carla Logging

The CARLA Python API offers a logger that saves certain values in an mmap file at a user-defined frequency. The logged values are as shown in 4.1.

4.1.2 MMap Decoder

Once the simulation is completed and the mmap file has been generated, it is processed using a custom Python script . This tool, from this point onward referred to as the MMap Decoder, converts the raw hexadecimal byte stream into a human-readable decimal representation, thereby enabling systematic analysis. Importantly, the MMap Decoder performs no modification of the underlying variable values; it only reformats their representation and storage structure. Specifically, it transforms the raw dictionary-based format into a structured CSV file that facilitates subsequent data handling, preprocessing, and model training. For this reason, its structure remains as shown in Table 4.1.

4.1.3 Data Preprocessing

The preprocessing stage is divided into 3 distinct parts. The first involves the use of the CARLA simulator to extract map-related information and the location of each vehicle. The second and third are performed independently of the simulator.

timestamp	index	quantity	mode_1	vehicle_id_1 \\
...	...	...	...	... \\
steer_1	throttle_1	brake_1	handbrake_1	reverse_1 \\
...	...	...	...	... \\
velocity_x_1	velocity_y_1	velocity_z_1	acceleration_x_1	acceleration_y_1 \\
...	...	...	...	... \\
acceleration_z_1	location_x_1	location_y_1	location_z_1	rotation_x_1 \\
...	...	...	...	... \\
rotation_y_1	rotation_z_1	....	rotation_y_41	rotation_z_41
...	...	...	...	...

Table 4.1. *MMap File Columns*

Custom Logger

For the first stage, an additional custom Python script was developed called Custom Logger. Its purpose is to process the output of the MMap Decoder and generate a new CSV file containing higher-level features, along with selected raw attributes carried over from the previous file.

Specifically, the timestamp, the rotational values around the three axes (yaw, pitch, roll), as well as the Lane ID, are transferred directly. Then, the signed instantaneous longitudinal and lateral velocities and accelerations are computed, under the convention that rightward motion is assigned a positive sign and leftward motion a negative one.

This computation is performed using the *get_forward_vector()*[56] function, the vector pointing forward according to the rotation of the object. By taking the cross product of these two vectors, the rightward lateral direction vector is obtained. With these three vectors, it becomes straightforward to transform the motion from the global Cartesian frame of the map to the local reference frame of each road segment, allowing for an accurate calculation of the longitudinal and lateral velocity and acceleration.

This transformation was a critical step in the preprocessing pipeline, as it ensured that the features used to train the model accurately captured the directional dynamics of vehicle motion within the local context of the road geometry.

timestamp	ego_vehicle_id	ego_lane_id	ego_yaw \\
...	...	...	... \\
ego_pitch	ego_roll	ego_lateral_-speed	ego_long_speed \\
...	...	...	... \\
ego_lateral_ac-celeration	ego_long_accel-eration	...	veh359_vehicle_-id \\
...	...	...	...\\
veh359_lane_id	veh359_yaw	veh359_pitch	veh359_roll \\
veh359_lateral_-speed	...	...	...\\
veh359_long_-speed	veh359_lateral_-acceleration	veh359_long_ac-celeration	
...	...	...	

Table 4.2. *CSV file columns after Custom Logger*

Data manipulation

Before proceeding to the second part of preprocessing, an exploratory data analysis was performed using visualizations to better understand the structure and meaning of the data. This analysis revealed that several rows contained missing values in columns of interest, which were removed as an initial step to ensure data integrity. The original CSV structure consisted of a timestamp in the first column, followed by the set of variables for each vehicle. Although straightforward, this representation was not well suited to our modeling objectives, as it grouped all vehicles into a single row per timestamp. To address this, the dataset was restructured such that each vehicle was assigned one row per data point. Consequently, the same timestamp could appear multiple times, once for each vehicle. To maintain continuity and avoid misalignment introduced by this stacking, we inserted 10 placeholder rows containing the value -10 at the end of each vehicle's sequence before appending the next vehicle's data. These artificial rows were later filtered out during input generation, serving as explicit markers of sequence boundaries.

Lane change detection was performed using the *Lane_ID* column. Through analysis, it was determined that when a vehicle moved from a lane with a higher absolute *Lane_-ID* to a lower one, it corresponded to a change in the left lane, while transitions from a lower to a higher absolute *Lane_ID* indicated a change in the right lane. Based on this rule, all *Lane_ID* values were first converted to their absolute form, after which

the difference between consecutive rows was computed. This transformation yielded the maneuver classes: -1 for left lane change, 0 for lane keeping, and 1 for right lane change. However, larger differences also appeared, which upon inspection were found to correspond to other maneuvers, such as highway entry or turns. These cases were systematically assigned the placeholder value -10, consistent with the sequence boundary convention. Finally, the values equal to -1 were replaced with 2, since the classes of the data to be classified must be positive integers.

This restructuring and labeling strategy was crucial, as it not only standardized the input format but also ensured robust identification of the maneuver classes of interest, while explicitly handling irrelevant or ambiguous events. As a result, the preprocessing pipeline produced a dataset that was both cleaner and better aligned with the specific requirements of the predictive modeling task.

timestamp	vehicle_id	lane_id	yaw	pitch \\
...	...	...	...	... \\
roll	lateral_speed	long_speed	lateral_acceleration	long_acceleration
...	...	...	...	...

Table 4.3. *Dataset columns after Data Manipulation*

Window Generation

The final step of the preprocessing stage was the construction of the model inputs. The features selected had to be values that could be obtained quickly and reliably in a real-world in-vehicle scenario. For this reason, we restricted the inputs to only two variables: the lateral velocity and the longitudinal velocity.

The model employs an LSTM layer as its input, which requires fixed-length time series as input sequences. To meet this requirement, sliding windows were created with the optimal window size found with grid search. Further examination of the training and evaluation datasets revealed that CARLA registers a lane change at the moment when the vehicle's center crosses the lane boundary. However, this point in time occurs too late to be considered an early prediction of the maneuver, which is the main goal. To overcome this limitation, a custom re-labeling strategy was developed in which timeseries slightly before and after the one originally labeled as a maneuver, were also labeled as one. The optimal timing of these before and after is also found with grid search. This strategy is also a novel way of exploiting CARLA simulation data to improve prediction performance.

During the creation of each window, we examine the values in the *Lane_ID* column from the start of the window up to 4 seconds afterward. If the value -10 appears or if the value corresponding to the opposite lane change occurs, that is, a value of 2 when generating windows for a lane change to the right or a value of 1 when generating windows for a lane change to the left, the window is discarded to reduce complexity.

4.1.4 Hyperparameter Tuning - Grid Search

Window Length

To determine the optimal window length, a grid search was performed. This brute-force technique searches through different hyperparameter values to identify the one that has the best overall performance. Specifically, five separate models were trained, each using a different window length. The performance of each model was then evaluated on the Real-World test dataset across multiple prediction horizons, and the configuration that achieved the best accuracy over different horizons was selected. The sequence lengths tested were 5, 7, 10, 12, and 14.

Finally, once the results of the next subsection are obtained, this grid search will be repeated to verify and confirm the consistency of the findings.

Temporal Offset Optimization of Timeseries

To determine how early, relative to CARLA's labeling of a lane change, a time series should be classified as a lane change maneuver, a second grid search was conducted. In this experiment, the window length identified in the previous grid search was kept constant, while the temporal offset between the CARLA-defined maneuver event and the labeling of a time window as a lane change was systematically varied. For each offset configuration, a new dataset was generated, where the sliding windows preceding the CARLA event by different time intervals were assigned to the corresponding maneuver class. Different models were then trained and tested for each created dataset. This procedure allowed for a detailed analysis of how the model's predictive performance changes as the labeling point moves further away from the moment the simulator officially registers the maneuver.

For better understanding, if in time $t_0 = 0$ CARLA registers a lane change, the following datasets will be created:

1. $[-1.4s, -0.2s]$
2. $[-1.4s, -0.2s]$, $[-1.2s, -0.0s]$
3. $[-1.6s, -0.4s]$, $[-1.4s, -0.2s]$
4. $[-1.6s, -0.4s]$, $[-1.4s, -0.2s]$, $[-1.2s, 0.0s]$
5. $[-1.8s, -0.6s]$, $[-1.6s, -0.4s]$, $[-1.4s, -0.2s]$
6. $[-1.8s, -0.6s]$, $[-1.6s, -0.4s]$, $[-1.4s, -0.2s]$, $[-1.2s, 0.0s]$
7. $[-2.0s, -0.8s]$, $[-1.8s, -0.6s]$, $[-1.6s, -0.4s]$, $[-1.4s, -0.2s]$
8. $[-2.0s, -0.8s]$, $[-1.8s, -0.6s]$, $[-1.6s, -0.4s]$, $[-1.4s, -0.2s]$, $[-1.2s, 0.0s]$

4.1.5 Training and Evaluation Data

To generate the training set we ran a CARLA simulation on Town04[57]. Town04 is a map that encompasses a variety of driving scenarios, as it features a figure-eight shaped highway surrounding a smaller urban road network composed of narrow one- and two-lane streets, interspersed with commercial and residential buildings. This map was selected because it is sufficiently large to accommodate a substantial number of vehicles—41 in this case—without causing traffic congestion that would hinder lane changes. Furthermore, its size helps to minimize the occurrence of repetitive data, meaning that the same maneuver is unlikely to take place multiple times at random locations.

To ensure that the data were as unique and non-repetitive as possible, at the start of the simulation and every 20 minutes thereafter, the following parameters were modified for all vehicles:

1. the probability of performing a lane change to the left,
2. the probability of performing a lane change to the right,
3. the individual speed limit.

The simulation lasted 6 hours and involved 41 vehicles. The data recording frequency was 5 Hz, which resulted in 102,404 data points per vehicle and a total of 4,198,564 data points. After applying the preprocessing steps described above, 48,233 inputs were generated.

- 11.564 right lane changes for the training and 5.064 for the evaluation set,
- 10.903 left lane changes for the training and 4.788 for the evaluation set,
- 11.067 lane keeping for the training and 4.847 for the evaluation set.

4.1.6 Testing Data

For the testing set, we wanted to use a different map, one with no relation to the previous environment, in order to avoid potential repetitions in the data. We selected the Town10[58] map, which represents a small urban environment characterized by rapid changes in orientation without lane changes, as well as numerous turns, making it suitable for testing our model.

The simulation lasted 85 minutes and included 11 vehicles. The data sampling rate was again set to 5 Hz, resulting in 25.112 data points per vehicle and a total of 276.232 data points. After applying the preprocessing steps described above with, the only difference being that a few more prediction horizons were added, 15.429 inputs were generated:

- 5.602 right lane changes,
- 4.799 left lane changes,
- 5.028 to lane keeping.

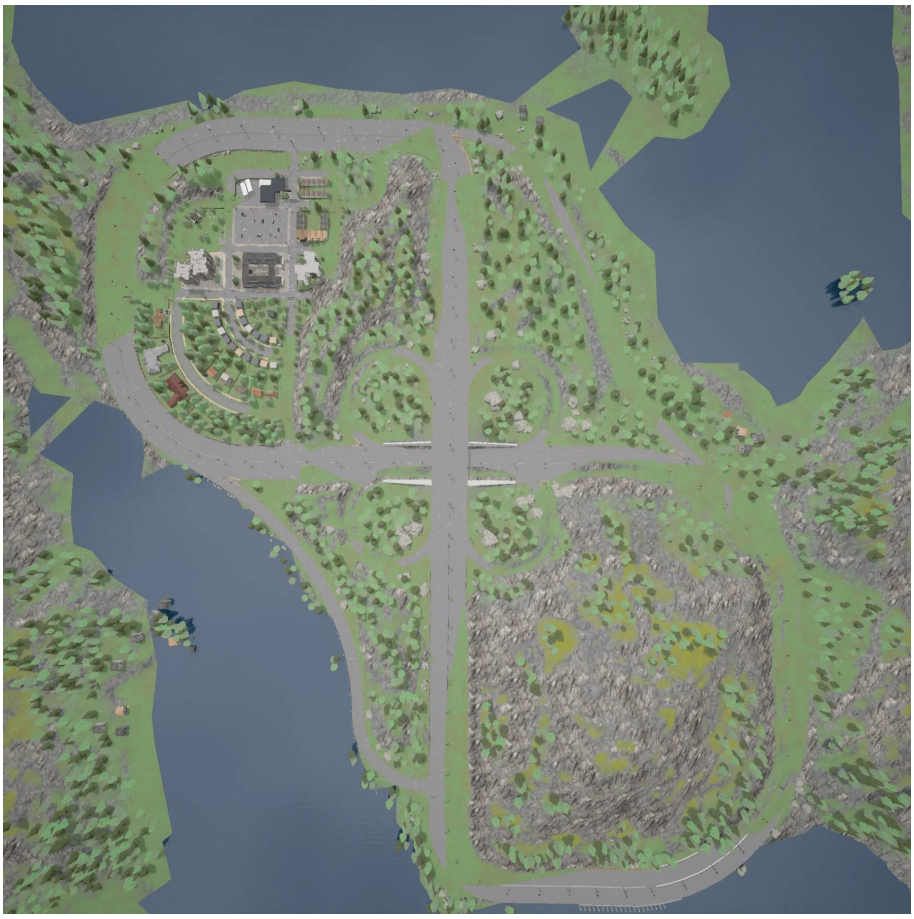

Figure 4.2. *Aerial view of Town04 map used for gathering training and evaluation data*

4.2 Model

As mentioned in the Theoretical Part at Section 3.3.5, a combination of LSTM Layers, that are better for capturing temporal features, and Fully Connected Layers for the classification part, were chose for the proposed Machine Learning Model.

1. LSTM with 256 units
2. Dropout Layer with 40% possibility
3. LSTM with 128 units
4. Dropout Layer with 40% possibility
5. Fully Connected Layer with 32 units and LeakyReLU activation function
6. Fully Connected Layer with 3 units and Softmax activation function

The architecture of the model is not very complex but it is complex enough to perform good in both Synthetic and Real-World data. A stacked LSTM was chosen instead of a single layer in order to capture both short-term signal variations and higher-level temporal patterns in the two velocity features. Dropout is applied after both LSTM layers to reduce

Figure 4.3. *Aerial view of Town10 map used for gathering testing data*

overfitting, which is a common issue with recurrent models. The Fully Connected layer with LeakyReLU helps the model separate features more effectively without running into the problem of inactive neurons. Finally, the Fully Connected with Softmax activation output layer provides probability scores for the three maneuver classes.

4.3 Training Process

The training process was relatively straightforward. First, the model described earlier was constructed and compiled using the Adam optimizer, categorical cross-entropy as the loss function, and accuracy as the evaluation metric, which is a common configuration for multi-class classification problems. Since the use of categorical cross-entropy as the loss function requires the labels to be one-hot encoded, the *to_ categorical* function provided by TensorFlow was used to perform this transformation. This transformation does affect the classes but just the representation of them. As a result, the labels now are as follows:

- 0 -> [1 0 0]
- 1 -> [0 1 0]
- 2 -> [0 0 1]

In addition, the training was designed to be as deterministic as possible, so that different training runs performed for hyperparameter tuning would not be influenced by randomness inherent in the learning process. To achieve this, several different random seed values were tested, and the one that yielded the best performance was fixed for all experiments.

Finally, an EarlyStopping callback was added to stop training when the validation accuracy failed to improve by at least 0.01 over 10 consecutive epochs.

Chapter 5

Methodology

5.1 Experimentation Scenarios

5.1.1 Testing with Synthetic Dataset

The first evaluation setup involved the use of synthetic data. After generating the data as previously described, the same preprocessing pipeline was applied, with the only modification being the addition of more prediction horizons, with those being: 1.0 second before the maneuver and 0.2 seconds after the maneuver. Consequently, our test dataset consists of timeseries that:

1. end 1.0 seconds before the maneuver
2. end 0.8 seconds before the maneuver
3. end 0.4 seconds before the maneuver
4. end 0.2 seconds before the maneuver
5. end 0.0 seconds before the maneuver
6. end 0.2 seconds after the maneuver

These prediction horizons were chosen because they are very close to the maneuver, and we considered it critical for the model to achieve very high accuracy for these intervals. Evaluating the performance of the model at these horizons allows us to decide whether to proceed with further assessment or make adjustments. A confusion matrix was then produced to obtain a comprehensive view of the model's behavior under these conditions.

Subsequently, the model was tested across different prediction horizons individually, in order to assess how its performance is affected by the temporal distance between the current observation window and the moment the maneuver is registered. To achieve this, all recorded lane changes were found , and sliding windows were created to end X seconds before the timestamp at which CARLA registered the maneuver. The values of X used were: 2.4, 2.2, 2.0, 1.8, 1.6, 1.4, 1.2, 1.0, 0.8, 0.6, 0.4, 0.2, 0.0, -0.2, -0.4, -0.6, -0.8.

5.1.2 Testing with Real-World Data

This research would be meaningless if there was no intention for it to also work in the real world. For this reason, we will attempt to test our trained model on real data as well. We will use data from the NGSIM US Highway 101 Dataset [17], which is quite well known and widely used in the literature[19][20][22], and from a recently released dataset collected from videos recorded with drones above the German Autobahn[18].

For both datasets, a confusion matrix was generated to provide an overall view for the prediction horizons of 1.0, 0.8, 0.6, 0.4, 0.2, 0, and –0.2 seconds. In addition, similarly to the tests conducted on the synthetic data, the model was evaluated across different temporal horizons using exactly the same procedure.

US Highway 101 Dataset

Starting with the US Highway 101 Dataset some preprocessing was required. First, we decided not to keep all the columns that were available, since many of them were not useful, but to keep only:

- Global_Time
- Vehicle_ID
- Total_Frames
- Local_X
- Local_Y
- Lane_ID

In the description of *Vehicle_ID* it is stated that repeated values are not associated. After analyzing the data, it was found that the combination of the columns *Vehicle_ID* and *Total_Frames* is that who uniquely identifies each vehicle. For properly processing the data, it should be known which record belongs to which vehicle. To achieve this, a new identifier, *New_Vehicle_ID*, was created for every record that has the same combination of *Vehicle_ID* and *Total_Frames*, and then the were data by this new identifier and *Global_Time*.

It was also observed that there were some temporal gaps, where no data were recorded for certain vehicles over a few seconds, which were not useful for our purposes. To detect these cases, a new column, *dt*, was created, which measured the difference in *Global_Time* between each record and the previous one. For each vehicle, the maximum *dt* value was checked, starting from the 3rd record. If this value was greater than 100ms, the entire vehicle was removed from the dataset. Vehicles with fewer than 15 records were also removed. Afterwards, as it was done with the synthetic dataset, 10 rows filled with the value -10 were added and *dt* column was dropped.

Another issue was that the sampling rate of the two datasets was dissimilar. The synthetic dataset had sampling rate of 5Hz, which means one sample every 1/5 of a second.

The real-world dataset, had sampling rate of 10Hz, which means one sample every 1/10 of a second. To make them consistent, only the odd indexed records were kept.

As already mentioned, the proposed model takes lateral and longitudinal speed as inputs. These values were not included in the first place, but could be calculated from the available columns. Specifically, *Global_ Time* corresponds to a Unix Timestamp (time in milliseconds since 01/01/1970 UTC). *Local_ X* is the lateral distance of the vehicle's front center from the leftmost section in the direction of travel, and *Local_ Y* is the longitudinal distance from the entrance of the section. By dividing the difference between two consecutive *Local_ X* values by the difference in their corresponding *Global_ Time* values the lateral speed was obtained, and by doing the same with *Local_ Y* gives the longitudinal speed. Finally, the results were multiplied by 0.30480 for the feet to meters conversion.

For *Lane_ ID* the same logic with the synthetic dataset was applied, but with one difference.In this dataset, the value 1 is a right lane change, while the value 2 is a left lane change.

German Autobahn Dataset

As with the previous dataset, the current data also required preprocessing to make suitable for this model. The dataset was split into two files, with each one corresponding to a different driving direction (West to East in one file and East to West in the other). The files were merged into a single dataset, keeping only the rows where the type of vehicle was Passenger Car.

As in the synthetic dataset, this dataset had a data recording frequency of 10 Hz. Downsampling was therefore applied again by keeping only the rows with odd indices. Columns that were not relevant for the modeling task, such as timestamps and vehicle dimensions, were removed, leaving only the variables necessary for model input:

- ID, which is a unique identifier for each vehicle
- x, which is the vertical position of the vehicle in space
- y, which is the horizontal position of the vehicle in space

In this dataset, lane identifiers were not provided. However, they were inferred using the y position, considering that the lanes on the Autobahn are 3.75 meters wide. By dividing y by 3.75, an artificial lane_id was created. A subsequent inspection of the distribution of y values, as shown in Figure 5.1, confirmed the validity of this approach, as vehicles tended to remain near $y = 1.9$ and $y = 5.8$, corresponding to the centers of the lanes. The same procedure for detecting lane changes as applied to the synthetic dataset was then followed.

5.2 Evaluation Metrics

For the evaluation of the model for each horizon, the following metrics will be used:

$$OverallAccuracy = \frac{TrueLLC + TrueRLC + TrueNLC}{TotalPredictions},$$

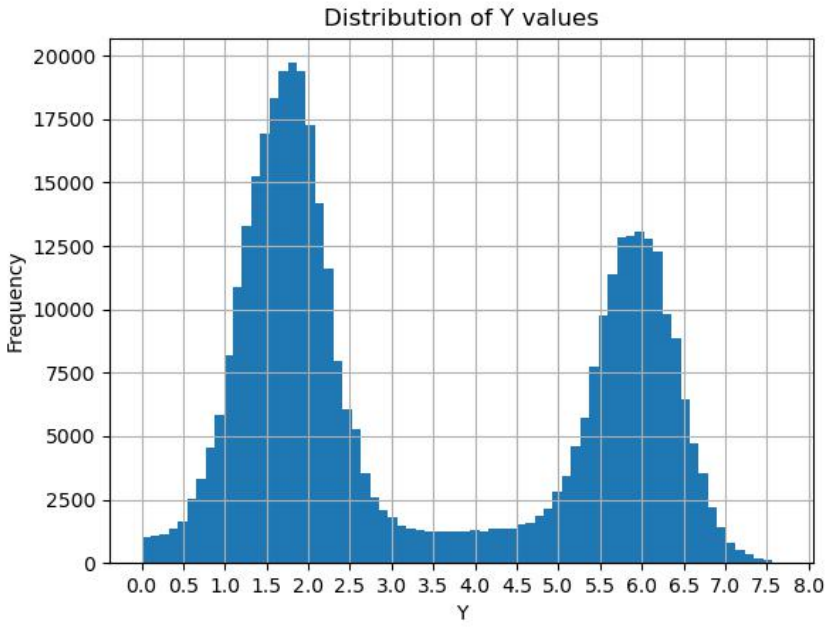

Figure 5.1. *Distribution of y values*

$$where \begin{cases} TrueRLC, & \text{Correctly predicted Right Lane Change} \\ TrueLLC, & \text{Correctly predicted Left Lane Change} \\ TrueNLC, & \text{Correctly predicted No Lane Change} \end{cases}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

5.3 Experimental Setup

The entire pipeline developed for this thesis was executed on a personal computer. The software environment consisted of Carla simulator version 0.9.14, PyCharm 2024.3.1.1 for the implementation of the Python scripts, MMap Decoder and Custom Logger, and managing the simulation configuration files, and Anaconda for Python package and environment management. TensorFlow, version 2.10.0, was employed as the primary machine learning framework and executed on the CPU. The preprocessing, model implementation, training, and evaluation were performed within Jupyter Notebook.

The hardware environment included a Ryzen 5600 processor, 16 GB of RAM, and an AMD RX 6700XT GPU.

Chapter 6

Results

Having thoroughly described the methodology in the previous chapter, we now proceed to present the results and their interpretation.

6.1 Hyperparameter Tuning Results

6.1.1 Window Length

After the execution of the first run of grid search to establish a solid baseline for the next experiment, the results shown in Figure 6.1 were obtained. As it can be observed, the best performance is achieved with a window length of 7, while a length of 5 also has competitive results.

Following the completion of 6.1.2, the first grid search was repeated using the final dataset. The results, shown in Figure 6.2, show a change in the optimal window length. The best overall performance is now achieved with a window length of 5, which will be used as the final configuration, instead of 7.

6.1.2 Optimization of Timeseries Labeling

Upon executing the second grid search, the results presented in Figure 6.3 were obtained. It is evident that the optimal performance occurs when for each lane change the time series which are labeled as one and added to the dataset are those following:

1. From $t_{start} = -2.0s$ to $t_{end} = -0.8s$ of Carla's Label
2. From $t_{start} = -1.8s$ to $t_{end} = -0.6s$ of Carla's Label
3. From $t_{start} = -1.6s$ to $t_{end} = -0.4s$ of Carla's Label
4. From $t_{start} = -1.4s$ to $t_{end} = -0.2s$ of Carla's Label
5. From $t_{start} = -1.2s$ to $t_{end} = 0.0s$ of Carla's Label

6.1.3 Final Values of Hyperparameters

In conclusion, the final value of Window Length is 5 timesteps or 1.0 second, while for the timeseries labeling, we label the following timeseries as a lane change:

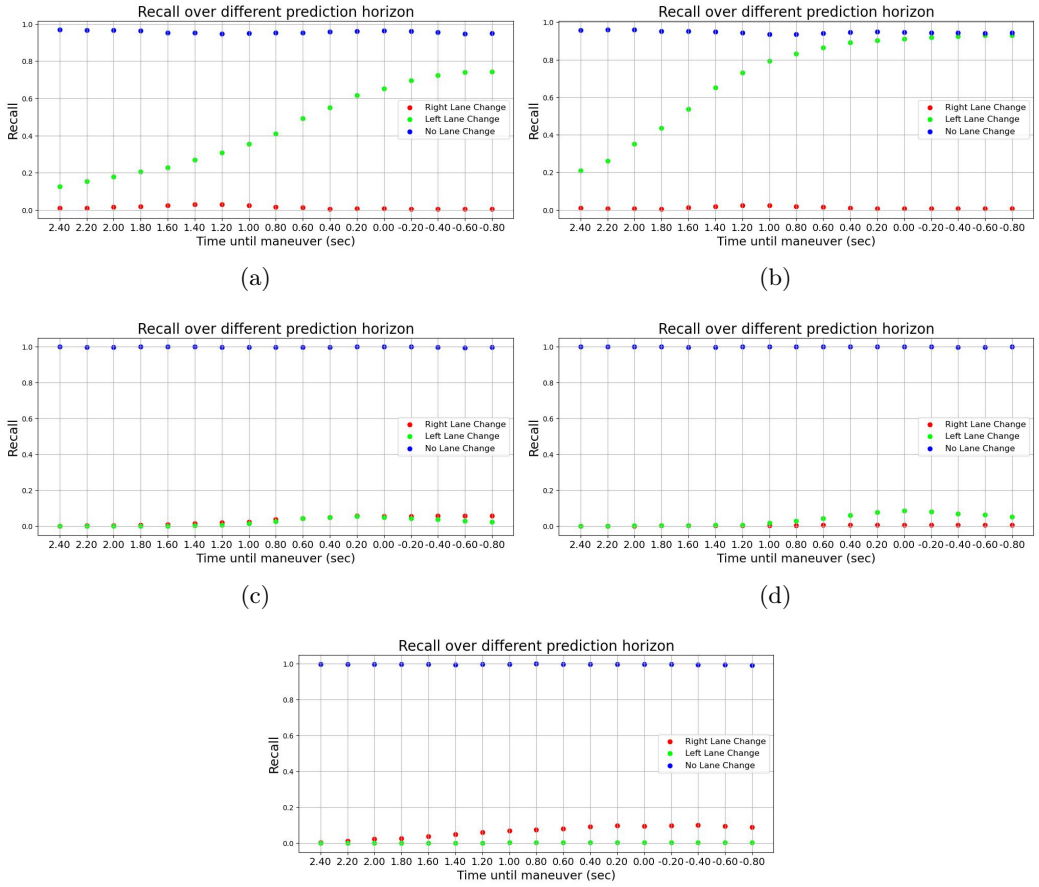

Figure 6.1. *Recall over different prediction horizons of models training with Window Length = (a) 5, (b) 7, (c) 10, (d) 12, (e) 14*

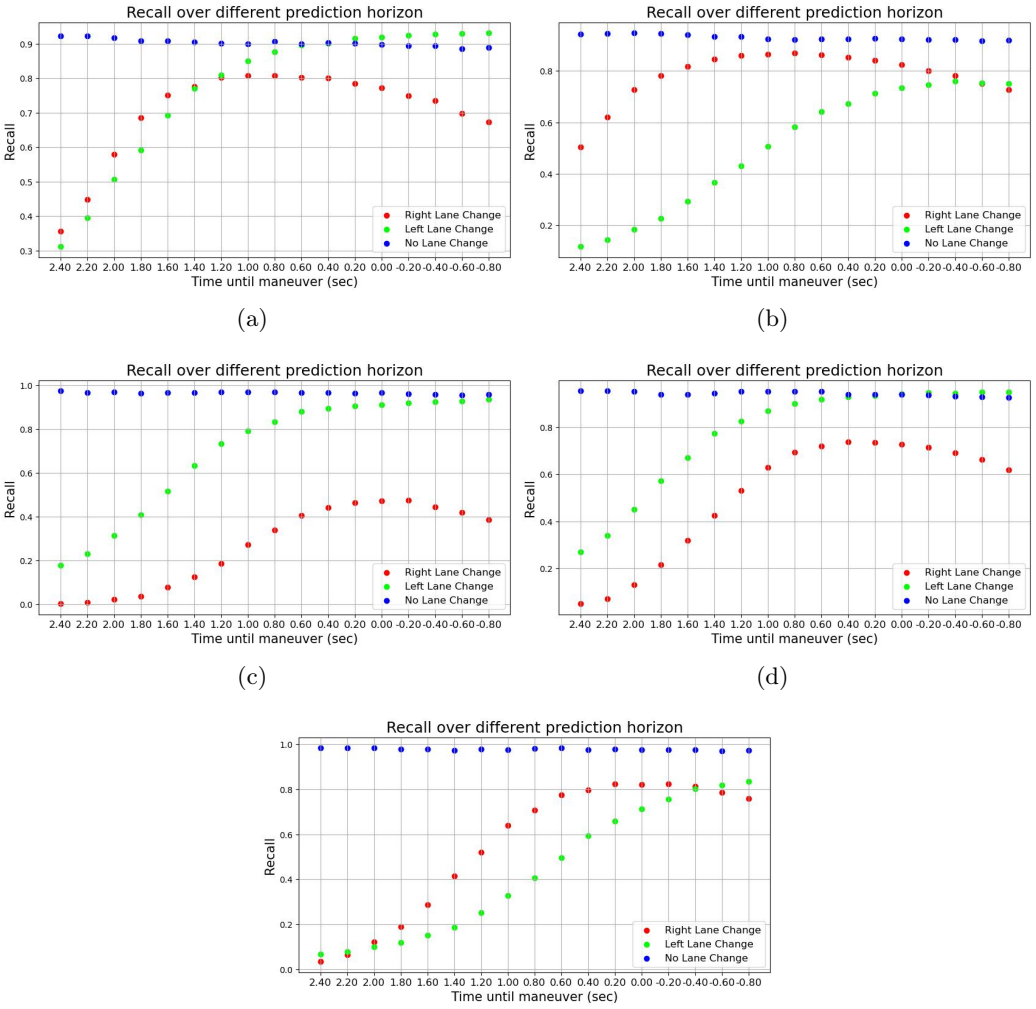

Figure 6.2. *Recall over different prediction horizons of models training with the final dataset and Window Length = (a) 5, (b) 7, (c) 10, (d) 12, (e) 14*

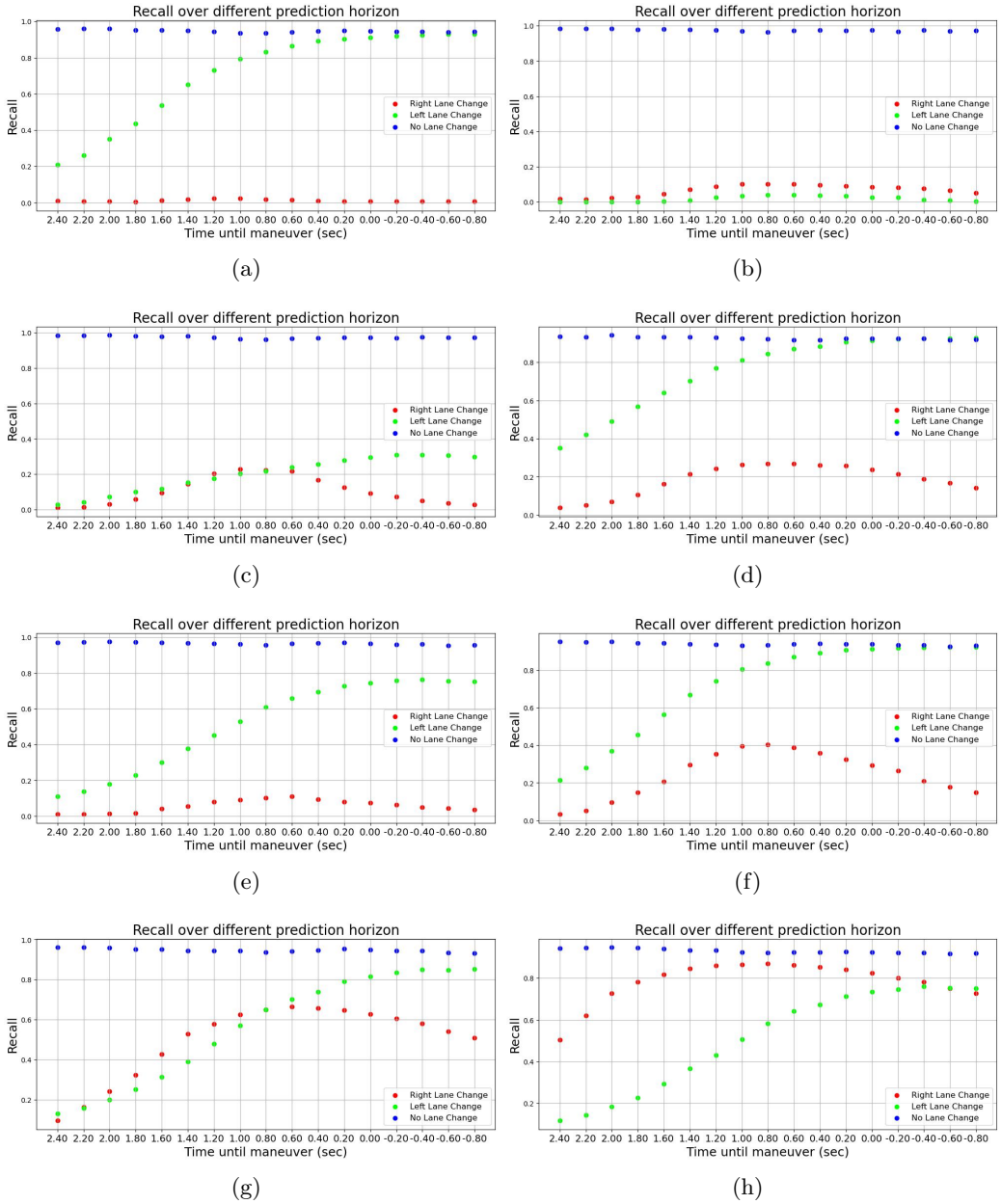

Figure 6.3. *Recall over different prediction horizons of models training with datasets as described in this list*

1. From $t_{start} = -1.6s$ to $t_{end} = -0.8s$ of Carla's Label
2. From $t_{start} = -1.4s$ to $t_{end} = -0.6s$ of Carla's Label
3. From $t_{start} = -1.2s$ to $t_{end} = -0.4s$ of Carla's Label
4. From $t_{start} = -1.0s$ to $t_{end} = -0.2s$ of Carla's Label
5. From $t_{start} = -0.8s$ to $t_{end} = 0.0s$ of Carla's Label

Figure 6.4 shows that 0.8 seconds, or four steps, before CARLA registers a lane change, the vehicle is less than one meter from the point where the lane crossing occurs. Considering that most standard vehicles are about two meters wide, it can be infered that at the moment we define as the earliest point to label a maneuver, the vehicle has already begun moving into the adjacent lane. This indicates that our choice for the earliest labeling is realistic and consistent with actual vehicle behavior.

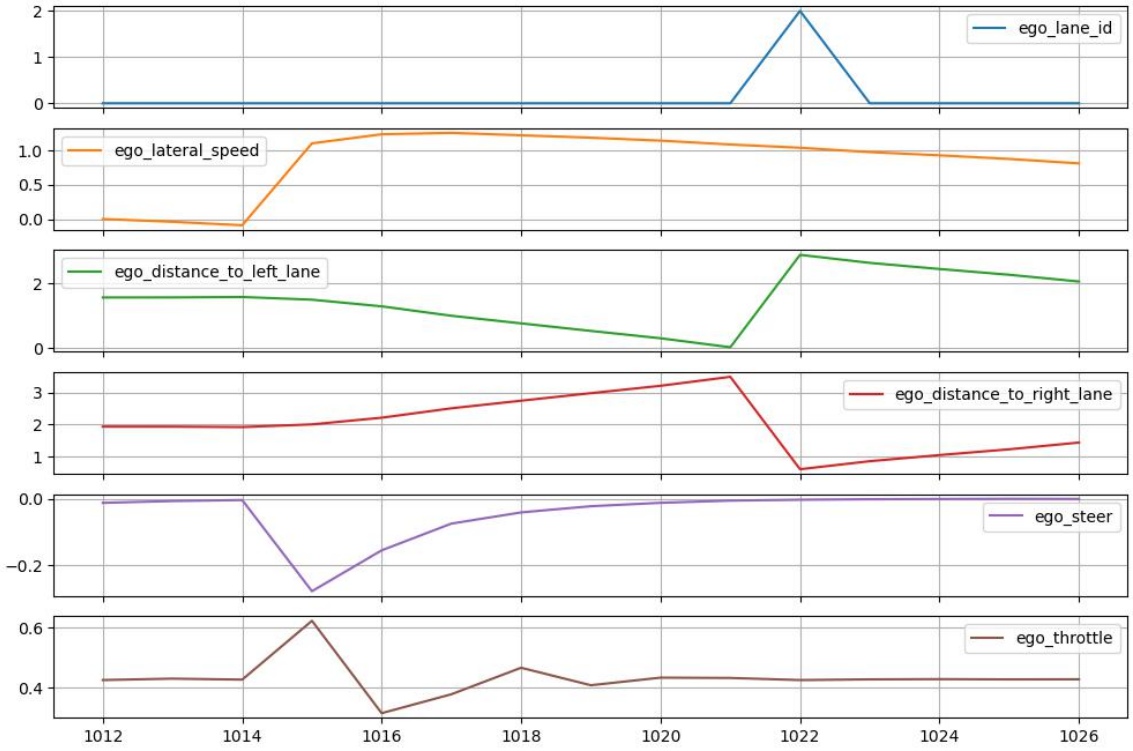

Figure 6.4. *CARLA data showing a Lane Change.*

6.2 Training Results

The model was trained for 16 epochs out of the 100 initially set, with the training stopped by the Early Stopping Callback. This callback was configured to stop training if validation accuracy did not improve by more than 0.01 for 10 consecutive epochs. After the first epoch, the model had already reached a validation overall accuracy of 94.3% with a validation loss of 0.2058. By the sixth epoch, accuracy increased to 95.3% and loss decreased to 0.1682, while the corresponding training metrics were 95.6% and 0.1569. No

improvement greater than 0.01 was observed after that point. The best performance was recorded in the 14th epoch, with a validation accuracy of 95.7% and a validation loss of 0.1537.

Figure 6.5 shows the evolution of accuracy and loss over the epochs.

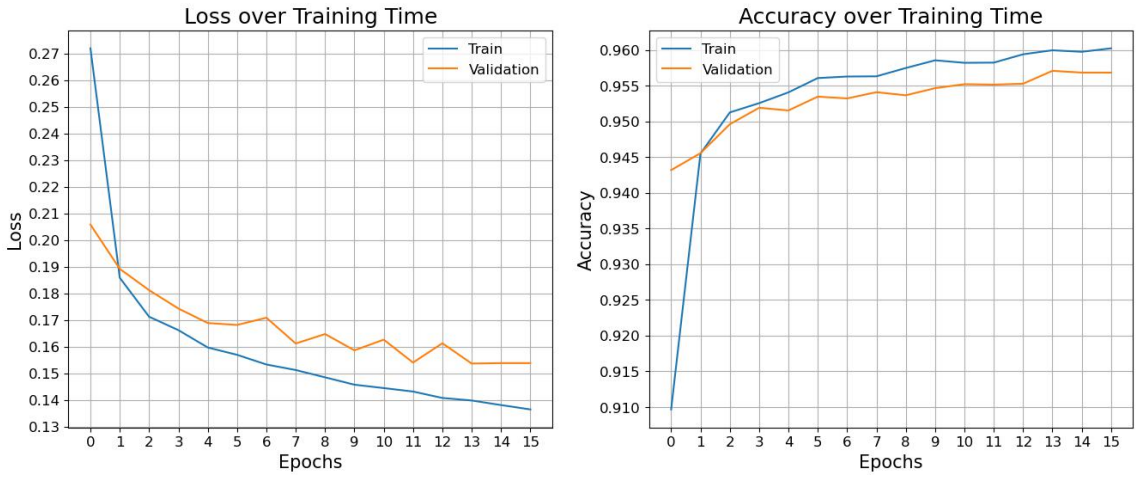

Figure 6.5. *Loss and Accuracy over Epochs*

To better understand why the validation accuracy was so high at the first epoch, a cosine distance analysis was performed between 200 validation samples and 2,000 training samples. The results showed that 75% of the samples were very similar (distance below 0.3), while only 5% differed significantly. This pattern was consistent across all splitting strategies.

For these reasons, the model's generalization ability will ultimately be evaluated using data from other simulation runs and, in particular, real-world data.

We subsequently trained the model without the Early Stopping Callback in order to assess its actual impact on the training process. As illustrated in the Figure 6.6 the model did not achieve any substantial improvement beyond the 14th epoch. From around the 20th epoch we see that Validation Loss increases while the Training Loss is decreasing which is a clear indication of Overfitting. The same stands for the Accuracy, where the training accuracy increases while the validation accuracy remains the about the same.

All of the above, justify the choice we made for the parameters of the Early Stopping Callback.

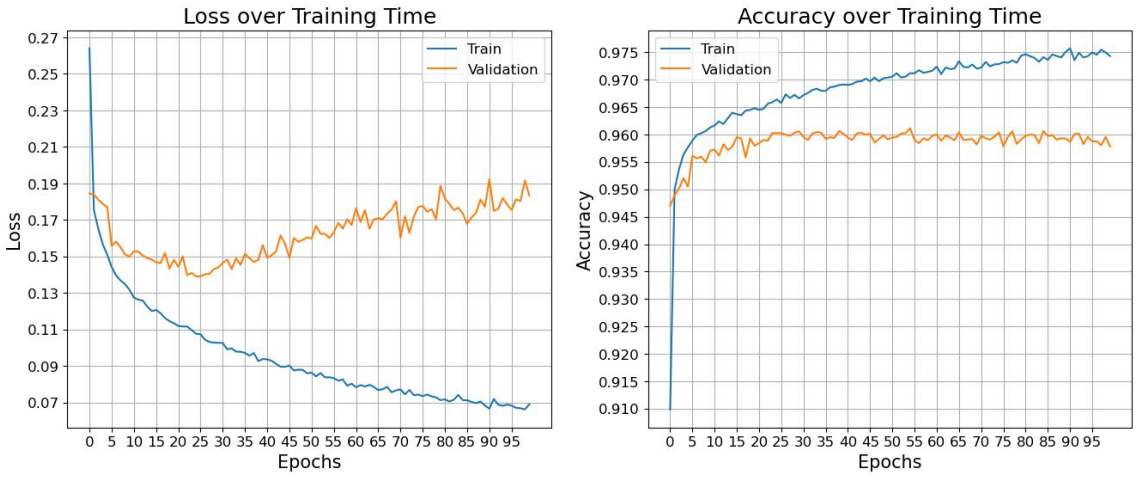

Figure 6.6. *Loss and Accuracy over Epochs Without Early Stopping Callback*

6.3 Testing with Synthetic Data Results

For the testing using synthetic data we first show a confusion matrix for the prediction horizons of 1.0, 0.8, 0.6, 0.4, 0.2, 0 and -0.2 seconds before the maneuver. As shown in Figure 6.7, the Recall the model achieved is 92.1% for lane keeping, 92.7% for right lane changes and 93.2% for left lane changes, with a total accuracy of 92.6%.

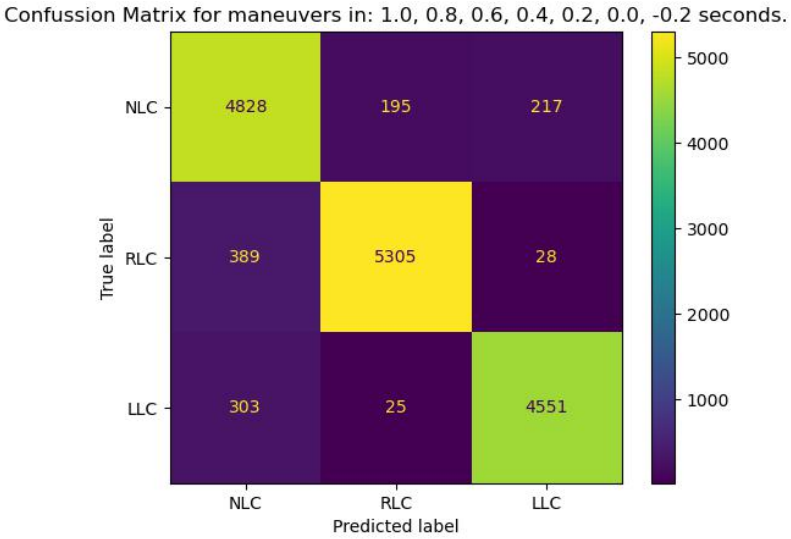

Figure 6.7. *Confusion Matrix for Synthetic Data*

Next, the four graphs with the metrics described in Section 5.2 are presented.

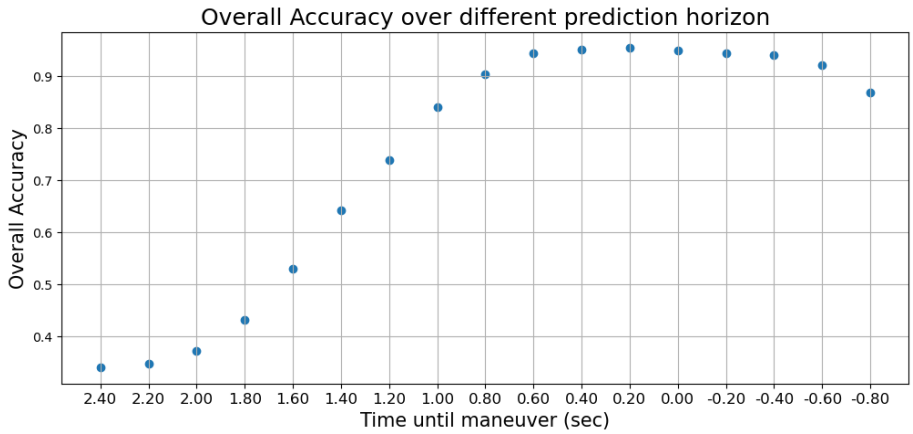

Figure 6.8. *Overall Accuracy over Different Prediction horizons*

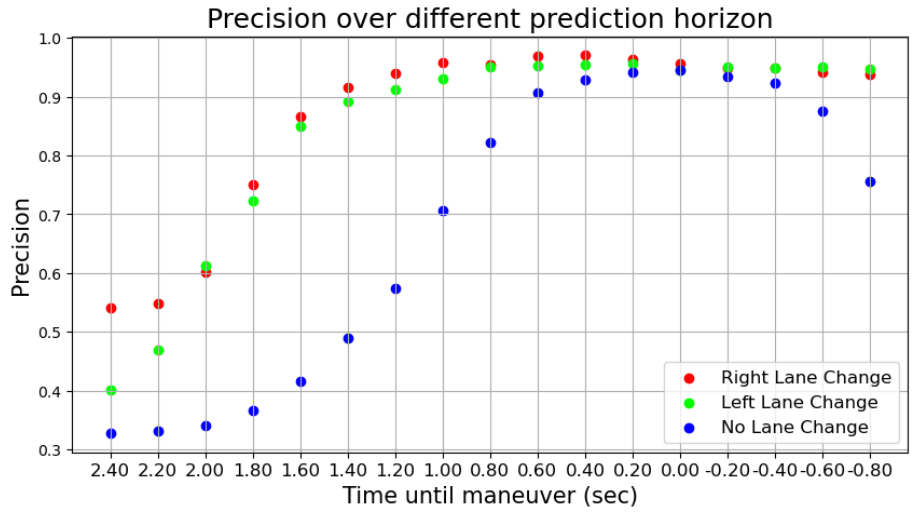

Figure 6.9. *Class-Wise Precision over Different Prediction horizons*

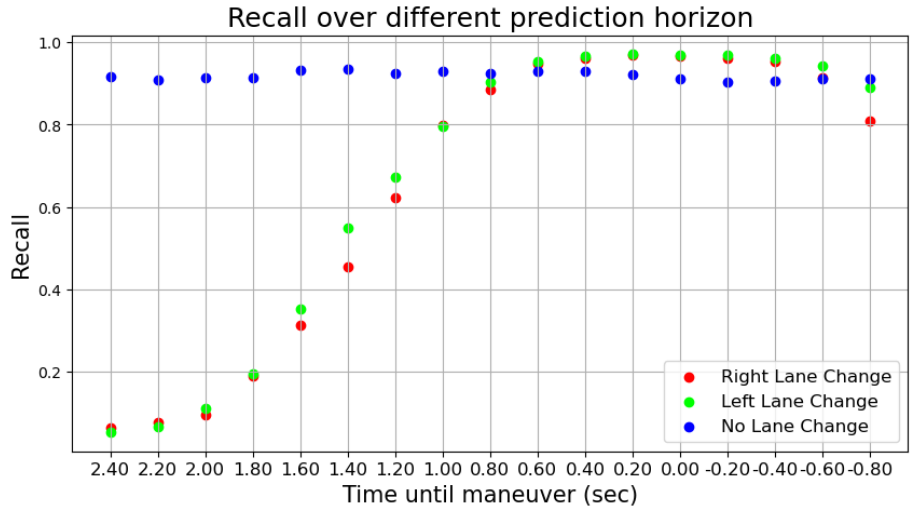

Figure 6.10. *Class-Wise Recall over Different Prediction horizons*

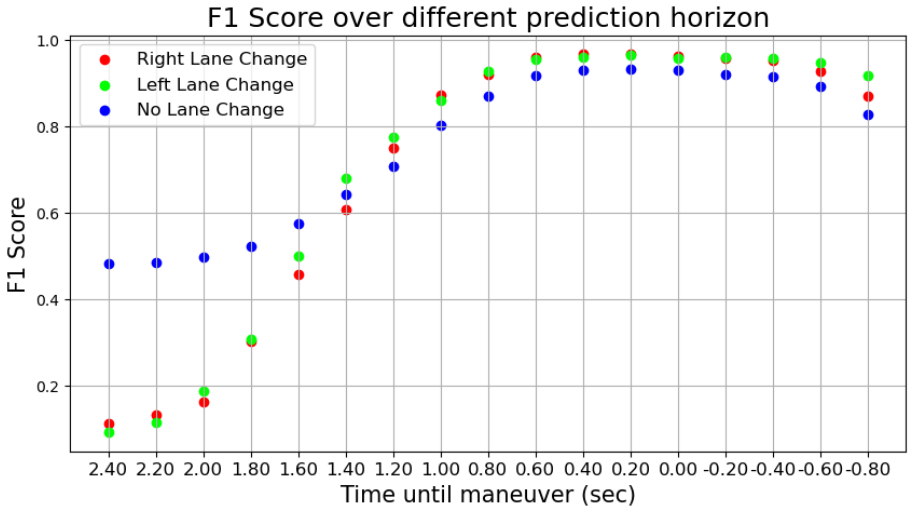

Figure 6.11. *Class-Wise F1-Score over Different Prediction horizons*

From the above graphs, it can be seen that the model was trained well and generalises effectively, even in scenarios it has never encountered before, such as the change from a highway map to an urban map. However, at large time distances from the maneuver (greater than 1.2 seconds), its performance drops sharply. On a good note though, the precision results show that the model is still able to correctly identify lane change maneuvers with relatively few false positives even at 1.8 seconds before the maneuver. This sh that, although overall confidence is lower at longer horizons, the model maintains a good level of reliability when it does predict a lane change, which is a desirable property for early warning systems.

In addition, the model was observed to exhibit a tendency to classify maneuvers as lane keeping rather than incorrectly predicting the direction of a lane change, a behavior that is particularly concerning. A closer examination of the data revealed that, within this time horizon, there are essentially no discernible indicators suggesting an upcoming lane change. This outcome is attributed to the high fidelity of the simulation: when a vehicle follows a straight trajectory, long before the actual decision to execute a maneuver is made, there are no small deviations in its motion that could signal the driver's intent.

As shown in Figure 6.12, up to point 1014, corresponding to 1.6 seconds before the maneuver is registered, no substantial variation is observed in either the vehicle's speed or its distance from the adjacent lanes. Minor oscillations appear in the steering and throttle signals, likely reflecting the activity of the PID controller as it attempts to maintain the vehicle at the center of the lane. Only at 1.4 seconds before the maneuver's registration does a clear lateral movement begin to emerge. This observation reinforces the earlier discussion regarding the observed decrease in accuracy. Figure 6.10 further illustrates this behavior, showing that model performance deteriorates sharply for time distances greater than 1.4 seconds before the maneuver.

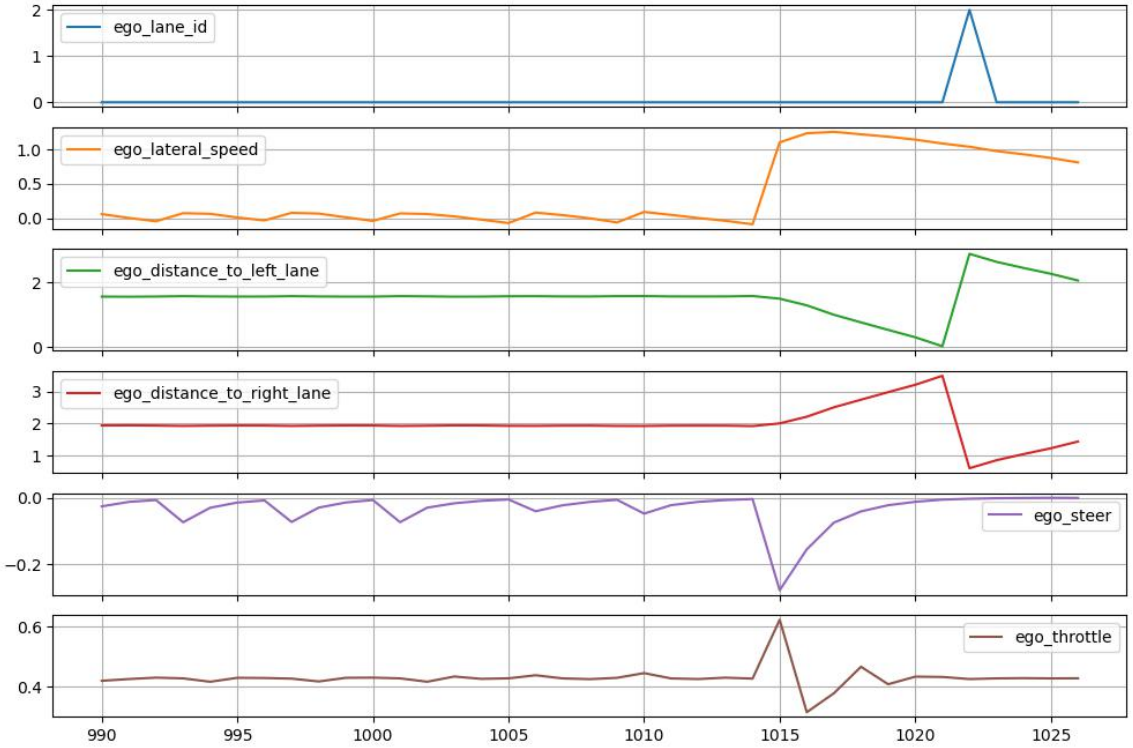

Figure 6.12. *Diagram of Lane_ID, Lateral Speed, Distance to Left and Right Lane, Steer and Throttle, showing a Left Lane Change. Diagram starts 6.4 seconds before the maneuver is registered.*

6.4 Testing with Real-World Results

6.4.1 US Highway 101 Dataset

As in the previous case, we tested our model using the prediction horizons of 1.0, 0.8, 0.6, 0.4, 0.2, 0 and -0.2 seconds before the maneuver and we first show the Confusion Matrix. The results are not very encouraging, even when considering that the model was trained exclusively on synthetic data. Specifically, the model achieved an accuracy of 94.3% for lane keeping, 56.5% for right lane changes, and 53.9% for left lane changes, with an overall accuracy of 69.1%. These results are further analyzed in Chapter 7.1.

Next, the four graphs with the metrics described in Section 5.2 over those prediction horizons we mentioned in Subsection 5.1.2 are presented.

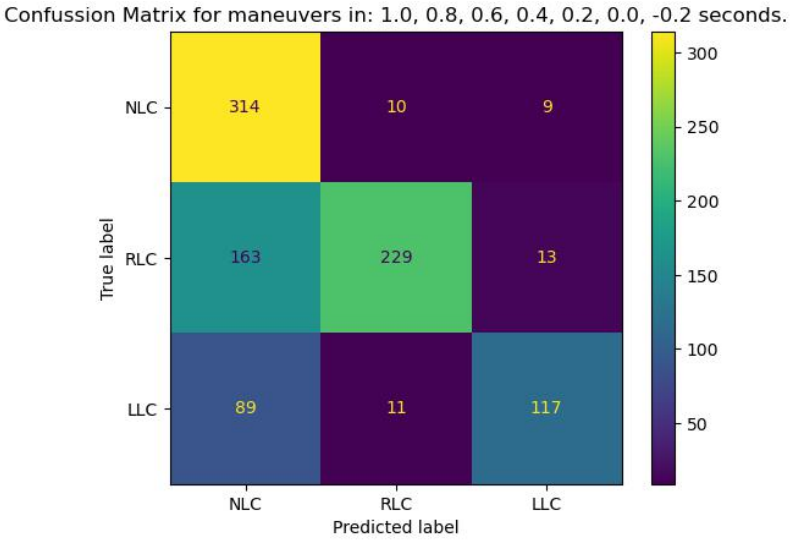

Figure 6.13. *Confusion Matrix for US Highway 101 Dataset*

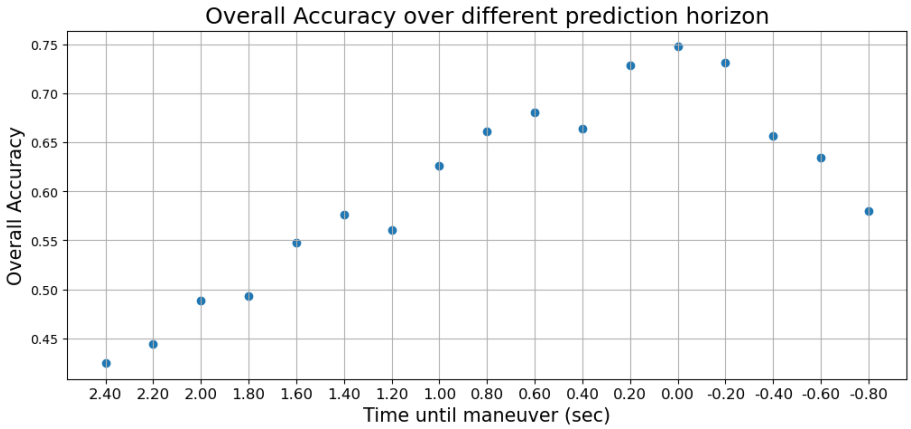

Figure 6.14. *Overall Accuracy over Different Prediction horizons*

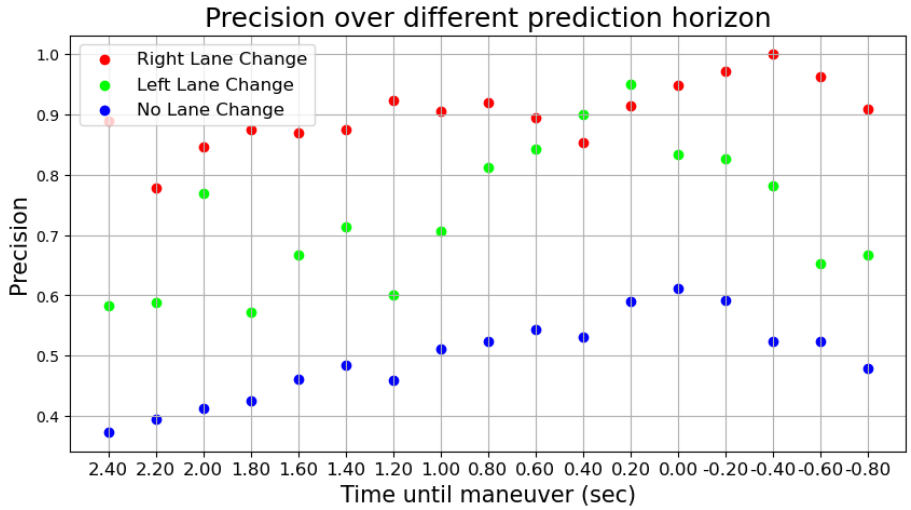

Figure 6.15. *Class-Wise Precision over Different Prediction horizons*

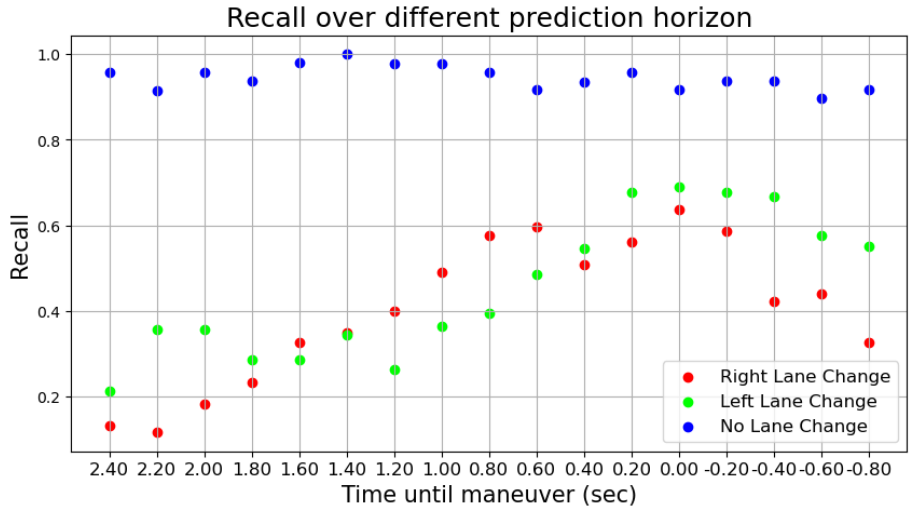

Figure 6.16. *Class-Wise Recall over Different Prediction horizons*

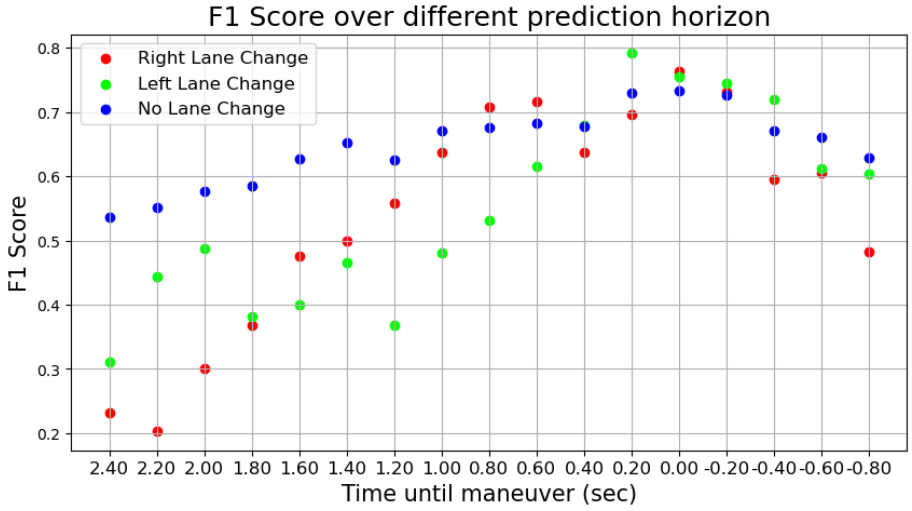

Figure 6.17. *Class-Wise F1-Score over Different Prediction horizons*

As already shown by the overall view of the confusion matrix, the model did not perform very well in several of the tested prediction horizons. As with the synthetic dataset, a positive aspect is that, as shown in Figure 6.15, it achieved a precision of higher than 60% in both lane-change classes. This means that when it predicts such a maneuver, it is unlikely to be wrong.

6.4.2 German Autobahn Dataset

As we did with the US Highway 101 Dataset, we first show the Confusion Matrix for prediction horizons of 1.0, 0.8, 0.6, 0.4, 0.2, 0 and -0.2 seconds before the maneuver. However, unlike with US Highwat 101 Dataset, this time the results are very encouraging, especially considering that the model was trained exclusively on synthetic data. Specifically, the model achieved an accuracy of 90.0% for lane keeping, 78.8% for right lane changes, and 89.7% for left lane changes, with an overall accuracy of 86.2%. These results are further analyzed in Chapter 7.1.

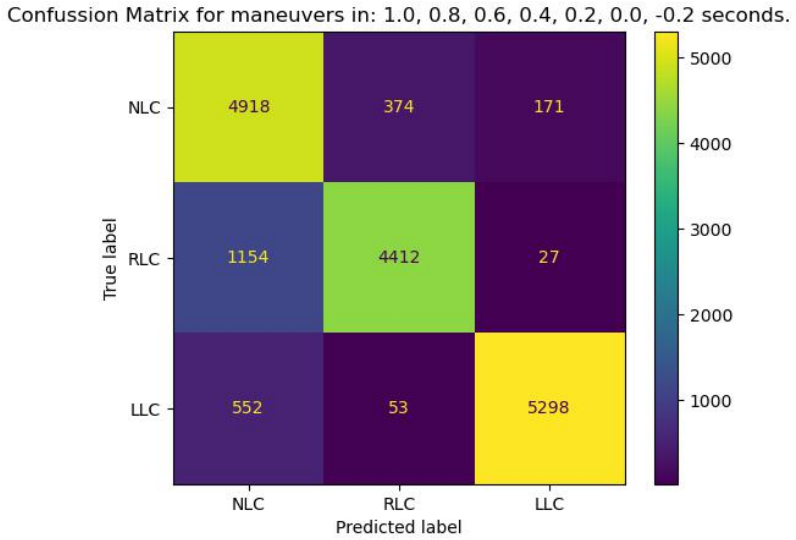

Figure 6.18. *Confusion Matrix for German Autobahn Dataset*

Next, the four graphs with the metrics described in Section 5.2 over those prediction horizons we mentioned in Subsection 5.1.2 are presented.

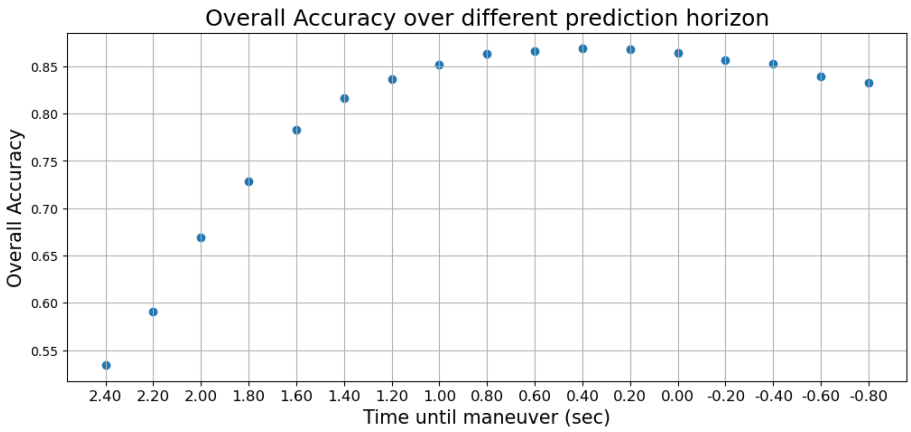

Figure 6.19. *Overall Accuracy over Different Prediction horizons*

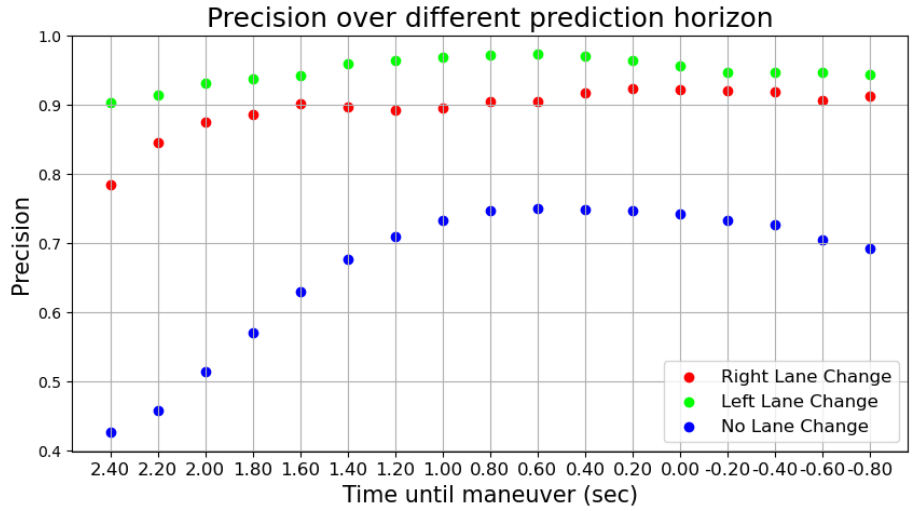

Figure 6.20. *Class-Wise Precision over Different Prediction horizons*

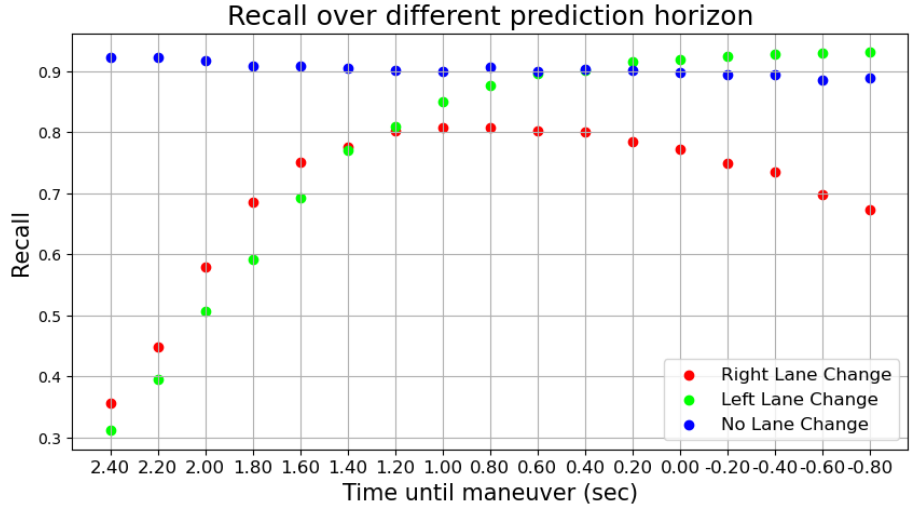

Figure 6.21. *Class-Wise Recall over Different Prediction horizons*

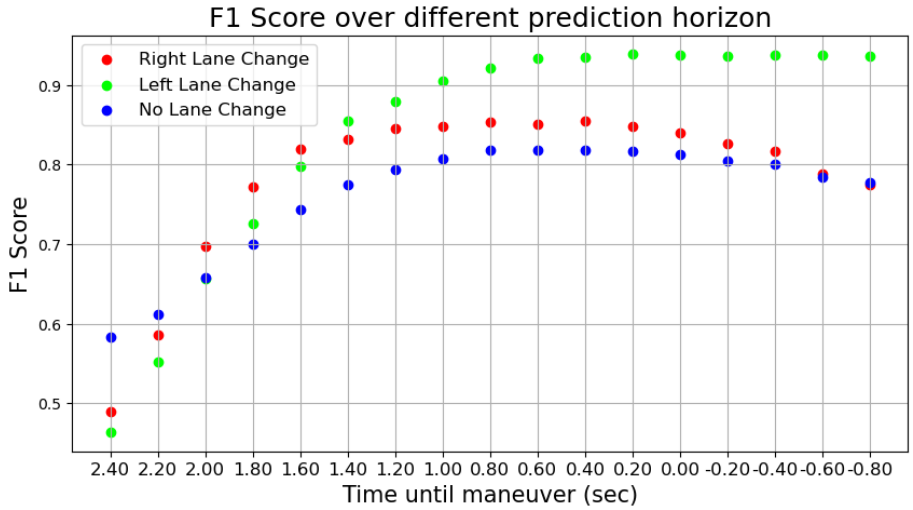

Figure 6.22. *Class-Wise F1-Score over Different Prediction horizons*

This time the results are very promising. Even at larger time distances of 1.8 seconds before the maneuver our model achieves 60% recall for Left lane changes and 70% for Right lane changes the model. The precision is very high even on larger prediction horizons reaching 85.0% for the Right Lane Change and 92.0% for the Left Lane change. Overall this is a very good and promising result.

6.5 Computanional Performance

The model was designed not only to achieve competitive predictive performance but also to remain computationally efficient, allowing deployment on systems with limited hardware capabilities. Both training and evaluation were performed on a CPU, as no GPU or TPU accelerator were available. Across all conducted tests—exceeding one million predictions—the average inference time per sample was 0.15 ms, indicating that the model can operate efficiently in real-time applications. This low inference latency enables GPU resources to be reserved for more computationally demanding tasks such as object detection.

The average duration of a training run, terminated by the Early Stopping Callback, was 8 minutes. During training, CPU power consumption was approximately 60 W, which is about 8Wh of energy per session. This value would likely differ if training were conducted using a GPU.

Part III

Epilogue

Chapter 7

Epilogue

7.1 Summary and Conclusions

In this thesis, we studied the problem of predicting vehicle behavior in urban environments using machine learning, focusing specifically on the prediction of imminent maneuvers through a machine learning algorithm trained on synthetic data generated by the CARLA simulator, a widely recognized tool in autonomous driving research. Subsequently, we evaluated the model's performance on synthetic data generated from a completely different map, changing from a highway scenario to an urban scenario, as well as, on real-world data from the NGSIM US Highway 101 Dataset[17] and the German Autobahn Dataset [18].

To achieve this objective, we first implemented a custom pipeline for the simulation data processing and use. We then ran a simulation on one map for 4.5 hours to generate the training dataset, and an additional 1.5 hours on the map to produce the testing dataset.

Next, we trained a relatively simple model, designed to ensure fast inference time. The training process lasted only ten minutes due to the simplicity of the model and the use of Early Stopping. For model evaluation, two different tests were executed using the generated data. In the first test we used data relatively close to the maneuver, with prediction horizons of 1.0 second as close as -0.2 seconds before the maneuver registration in CARLA. In the second test, we calculated four different metrics for more prediction horizons from 2.4 before the maneuver up to 0.8 seconds after the maneuver.

The results of the first scenarios were highly encouraging, with the model achieving overall prediction accuracy of up to 92.6%. The second test better pictured the true performance of the model. Up until prediction horizons of 1.0 second it performs very well. However, for larger time distances it produced less satisfactory results, falling about 10% for each 0.2 seconds behind in time. This is justified because a maneuver begins at the moment a decision is made by the CARLA Traffic Manager, and until that point, the vehicle maintains its position at the center of its current lane. This decision is randomly determined by the Motion Planner based on a predefined probability. Subsequently, a PID controller generates the control commands, which the vehicle then executes.

To further illustrate this, in Figure 7.1 a sample lane-change event is presented to help interpret the obtained results. The graph begins 15 data points before the lane-change registration and ends 10 data points after. Recall that a lane change is registered when

the vehicle's center is exactly aligned with the dividing line between two adjacent lanes. Consequently, the graph spans 3 seconds prior to the registration point and 2 seconds after. We observe that the vehicle starts to increase its lateral velocity at $t = 2.0s$, which corresponds to just 1 second before the lane change is registered, and returns to zero at $t = 4.8s$, or 1.8 seconds after. This shows that the maneuver is rather abrupt and aggressive, which in turn explains why our model struggles to make accurate predictions earlier than 1 second before the registration point.

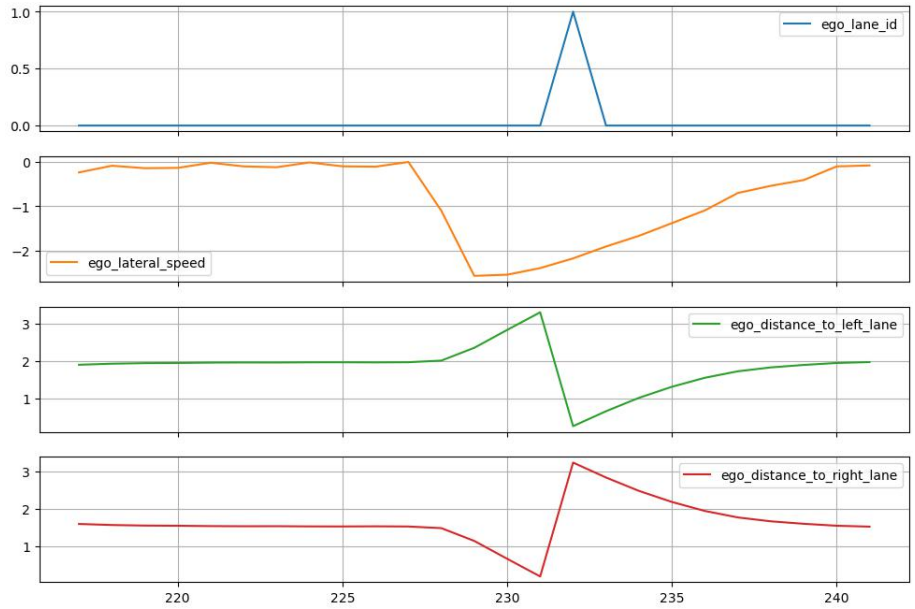

Figure 7.1. *A graph showing the values of Lane ID, Lateral Speed, Distance to Left Lane, Distance to Right Lane during a lane change to the right*

The final step was to conduct a more substantial evaluation of our model, namely by testing it on real-world data from the dataset [17] and [18]. For the first dataset the results, while satisfactory, did not fully meet our expectations, particularly in the case of left-lane changes. In order to better understand these outcomes, we proceeded with an analysis of the dataset.

We were able to classify the mispredictions into three categories. The first category included cases where the lane change was performed very slowly, that is, with relatively low lateral velocity. By artificially multiplying the lateral velocity by a factor of two or even four, we observed that the model's prediction changed accordingly and became correct. The values of *Lane_ID*, *Lateral speed(U_X)* and the *Distance from the left-most section of the road (Local_X)* of such cases are shown at Figure 7.2.

Another category of issues was related to corrupted data. Since this dataset was created in 2005 and extracted from video recordings, errors are to be expected. The most common error was the incorrect assignment of vehicles to lanes. For example, in one case a vehicle was recorded as performing a lane change when its *Local_X* value (the distance of the vehicle from the left edge of the road) was 15 meters. The vehicle then continued its trajectory up to a value of 24 without any lane change being annotated, later dropped to 17, and finally a lane change to the right was registered at 19 meters. In another case

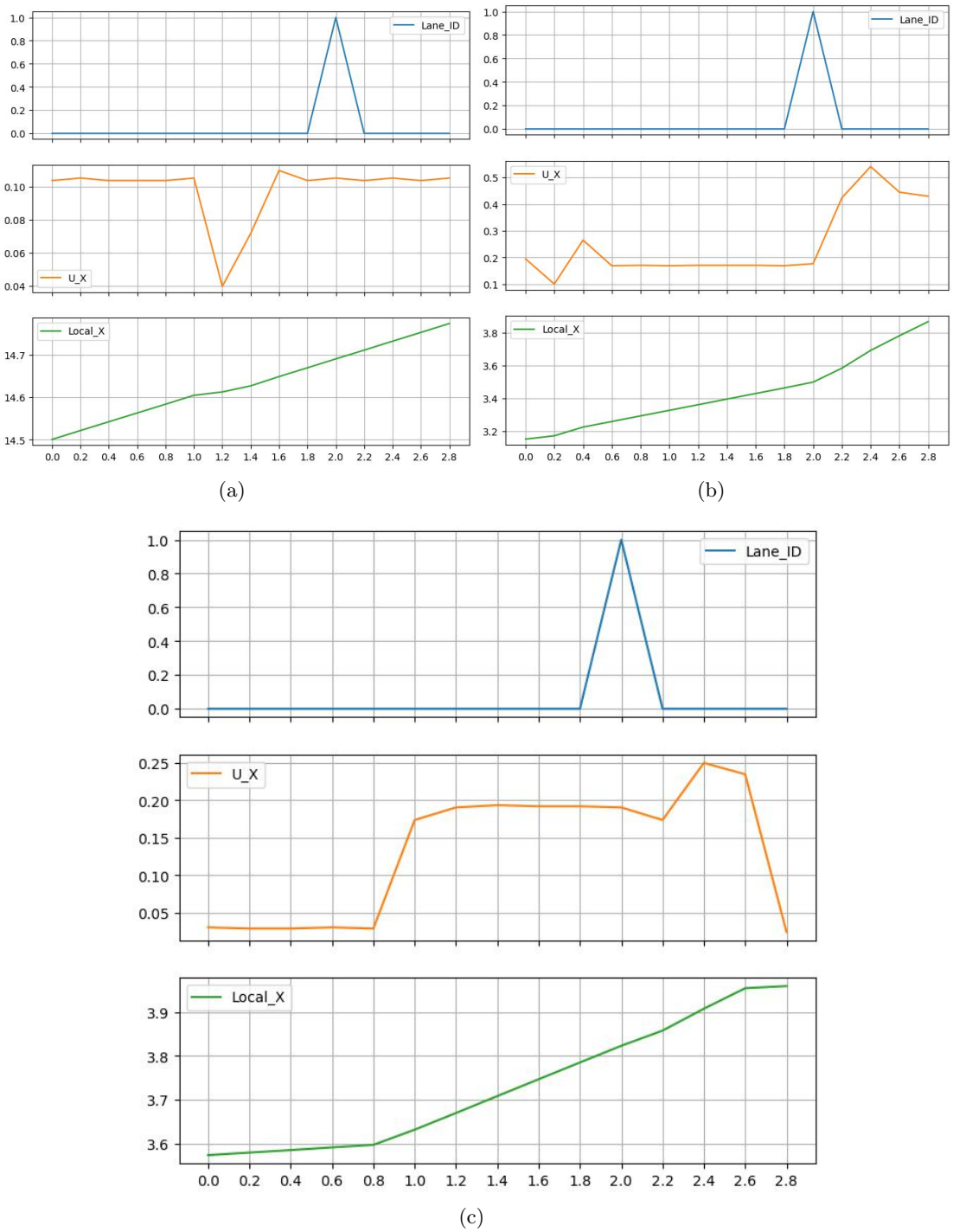

Figure 7.2. *Graphs of three incorrectly categorized lane changes possibly due to low speed.*

shown at Figure 7.3, a vehicle was annotated as making two consecutive right lane changes, while in both instances its Local_X value was around 6.5 meters, without any preceding lane change to the left.

The final category consisted of cases for which we were unable to identify a clear reason for the misclassification. Most of the data belonging to this category contained rows with a sign opposite to what would be expected for the corresponding maneuver, which we consider may have contributed to the incorrect predictions.

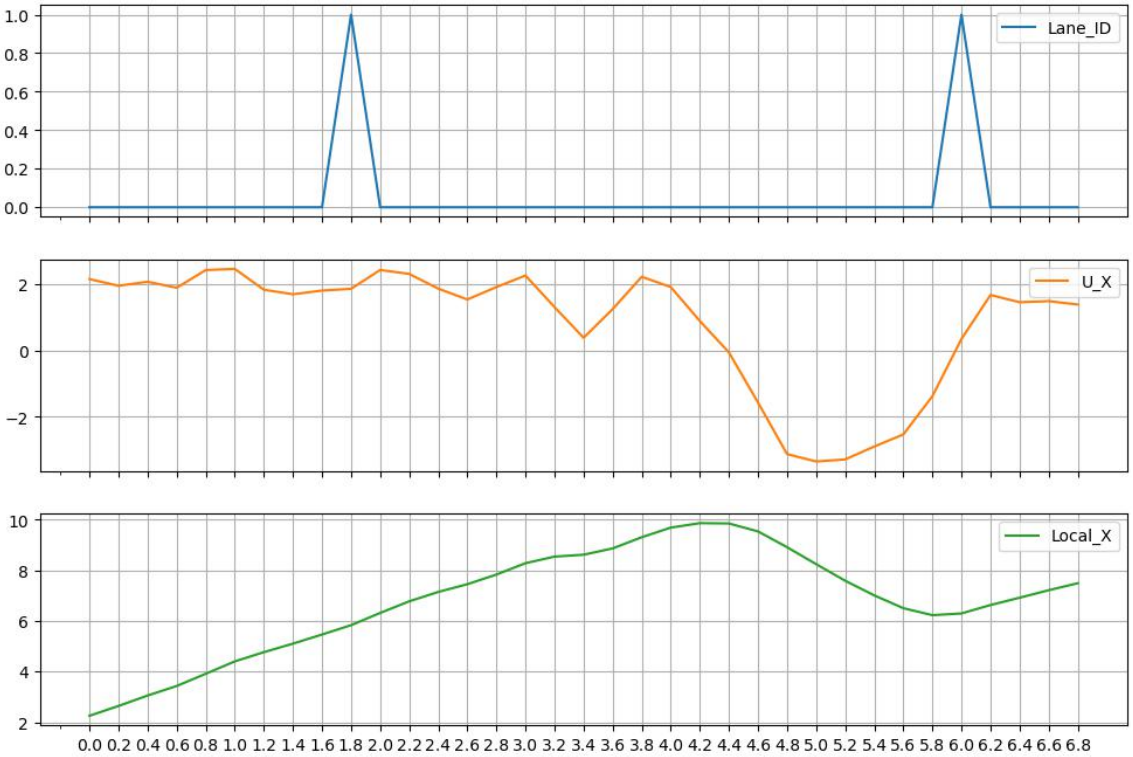

Figure 7.3. *Corrupted Real-World Data*

Continuing with the real data and the German Autobahn Dataset, the results were noticeably better than those of the US Highway 101 Dataset and, in some cases, even better than the synthetic data. Specifically, the overall accuracy remained high even 1.8 seconds before the maneuver, reaching 75%, while the precision stayed at 80% even at 2.40 seconds before the maneuver.

Based on these results, two conclusions can be drawn. The first is that training machine learning models with synthetic data and then applying them in real conditions is feasible. The second is the confirmation that creating a reliable model that performs well globally is quite challenging and requires a large amount of data. As shown, the same model, in the same type of environment—highways with long straight sections—but in a different country with different rules, did not perform equally well in both scenarios.

These findings have broader implications for the field of autonomous driving. Successfully training models on synthetic data and transferring them to real-world scenarios opens avenues for efficient development and testing of predictive systems, reducing the reliance on costly and time-consuming real-world data collection. Understanding the limitations

of these models and the conditions under which they perform best can help guide the development of future autonomous systems, improving the reliability of vehicle behavior predictions in complex urban settings.

7.1.1 Challenges and Limitations

As expected, the development of this work had some limitations and challenges. One category of limitations was the experimental configurations that could not be implemented, either because they fell outside of the scope of this thesis or because they were technically impossible. For instance, adjusting the PID controller parameters individually for each vehicle was initially considered, but CARLA only allows modifications to these parameters collectively across all vehicles. Another intended configuration was the modeling of the impact of weather and road conditions on vehicle traction. Although different weather and road states were tested, they did not result in any noticeable changes in vehicle behavior, and therefore this aspect could not be effectively incorporated.

Another challenge was the reproducibility of the experimental results. During the first training attempts, no random seed had been set, and while one training run produced decent results, the next run under the same configuration produced worse performance due to randomness. This highlighted the importance of controlled initialization when evaluating model performance.

Another limitation observed was the weakness of CARLA to handle large number of vehicles, especially at high speeds. Initial simulation runs included 100 vehicles on fairly large maps, which could accommodate this number. However, increasing the number of vehicles led to frequent collisions that completely blocked traffic over large areas of the map. Using 40 vehicles, which we determined to be a reasonable number, did not create this problem.

A further limitation is the simulation speed. Ideally, we did not want the ratio between real time and simulation time to be 1 to 1, but the simulation to run faster than real time. CARLA does not provide a straightforward way to achieve this, and the methods we tested either had no effect or caused the simulation to fail. While a solution may exist, we did not pursue it further after several unsuccessful attempts.

Finally, a limitation we encountered was related to our hardware setup. The combination of an AMD GPU with Windows proved incompatible for machine learning due to missing drivers, preventing the GPU from functioning as an accelerator. This limitation prevented us from testing larger and deeper architectures that required more resources and also restricted the parallel use of the system for long periods of time.

7.2 Future Work and Extensions

The present work is open to several roads for future research, and we outline here the most noteworthy.

The first to consider would be the intergration of the developed model directly into the CARLA simulator, enabling real-time predictions for surrounding vehicles relative to

the ego vehicle. This extension would require a better understanding of CARLA's internal processes to allow the parallel execution of additional tasks, as well as the development of an intergrated data pipeline. Instead of relying on two external Python scripts, the selected features could be passed directly to the model in real time. In addition, deploying the system on hardware equipped with NVIDIA GPUs could allow faster inference times and make the approach more suitable for real-world scenarios.

Another extension would be the modification of the Traffic Manager in order to achieve bigger variety of vehicle driving behavior. By changing the parameters of the PID controllers, it would be possible to simulate vehicles with different lane change strategies, with some being more aggressive with abrupt maneuvers, while others do smoother and more gradual transitions. Having more behavioral diversity could lead to more realistic synthetic datasets and, consequently, to models with stronger generalization capabilities.

Beyond maneuver classification, future work could also include trajectory prediction, as reviewed in Section 2.1. This would be training models to predict the exact trajectory a vehicle is likely to follow. Doing such research with synthetic data and then testing them with real-world datasets would represent an important step toward bridging the gap between simulation and reality.

Finally, an interesting work would be an in-depth comparative study of different model architecture proposed in the literature, tested with datasets generated with CARLA. This kind of work could provide insights into which approaches are best suited for learning from synthetic driving data, and guide the design of more robust predictive models.

Appendices

Appendix A

Source Code

The complete source code used in this thesis, including the dataset preprocessing pipeline and the machine learning models, is available in a public GitHub repository. The repository contains scripts for data preprocessing and windowing of time-series data, and training and evaluation of LSTM-based models for vehicle behavior prediction.

The code can be accessed at: https://github.com/FoxyStent/carla-lane-change-prediction

Bibliography

[1] SAE International. *SAE Levels of Driving Automation™ Refined for Clarity and International Audience*, 2021. Accessed: 2025-09-10.

[2] Haoxuan Luo, Xiao Hu και Linyu Huang. *A Hybrid Model for Vehicle Acceleration Prediction*. *Sensors*, 23(16), 2023.

[3] GeeksforGeeks. *Classification vs Regression in Machine Learning*, 2025. Accessed: 2025-07-14.

[4] Mamunur Rahaman, Chen Li, Yudong Yao, Frank Kulwa, Mohammad Rahman, Qian Wang, Shouliang Qi, Fanjie Kong, Xuemin Zhu και Xin Zhao. *Identification of COVID-19 samples from chest X-Ray images using deep learning: A comparison of transfer learning approaches*. *Journal of X-ray science and technology*, 28, 2020.

[5] Junxi Feng, Xiaohai He, Qizhi Teng, Chao Ren, Honggang Chen και Yang Li. *Reconstruction of porous media from extremely limited information using conditional generative adversarial networks*. *Physical Review E*, 100, 2019.

[6] Nadia Nedjah, Igor Santos και Luiza Mourelle. *Sentiment analysis using convolutional neural network via word embeddings*. *Evolutionary Intelligence*, 15, 2019.

[7] İlyürek Kılıç. *Perceptron Model: The Foundation of Neural Networks*, 2023. Accessed: 2025-07-16.

[8] Mike Moore. *What is Industry 4.0? Everything you need to know*, 2020. Accessed: 2025-10-06.

[9] G Velasco και J P Schnell. *Gas sensors and their applications in the automotive industry*. *Journal of Physics E: Scientific Instruments*, 16(10):973, 1983.

[10] *Airbag Technology: What It Is and How It Came to Be (980648)*, σελίδες 53–72. SAE, 2000.

[11] Mercedes Benz. *Mercedes Benz Parktronic system (PTS)*, 2004. Accessed: 2025-10-06.

[12] S J Prosser. *Automotive sensors: past, present and future*. *Journal of Physics: Conference Series*, 76(1):012001, 2007.

[13] Data Bridge Market Research. *Global Automotive Sensors Market Size, Share, and Trends Analysis Report – Industry Overview and Forecast to 2032*, 2025. Accessed: 2025-10-08.

[14] Mercedes Benz. *Support speed of up to 95 km/h on German motorways.*, 2024. Accessed: 2025-10-06.

[15] U.S Department Of Transportation. National Highway Traffic Safety Administration. *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*, 2015. Accessed: 2025-10-06.

[16] Tesla. *Full Self-Driving (Supervised)*. Accessed: 2025-10-06.

[17] U.S Department Of Transportation. Federal Highway Administration. *US Highway 101 Dataset*, 2007.

[18] Moritz Berghaus, Serge Lamberty, Jörg Ehlers, Eszter Kalló και Markus Oeser. *Vehicle trajectory dataset from drone videos including off-ramp and congested traffic – Analysis of data quality, traffic flow, and accident risk*. *Communications in Transportation Research*, 4:100133, 2024.

[19] Florent Altché και Arnaudde La Fortelle. *An LSTM network for highway trajectory prediction*. *2017 IEEE 20th international conference on intelligent transportation systems (ITSC)*, σελίδες 353–359. IEEE, 2017.

[20] Mahrokh Khakzar, Andry Rakotonirainy, Andy Bond και Sepehr G Dehkordi. *A dual learning model for vehicle trajectory prediction*. *IEEE Access*, 8:21897–21908, 2020.

[21] Robert Krajewski, Julian Bock, Laurent Kloeker και Lutz Eckstein. *The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems*. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, σελίδες 2118–2125, 2018.

[22] Zihao Sheng, Yunwen Xu, Shibei Xue και Dewei Li. *Graph-based spatial-temporal convolutional network for vehicle trajectory prediction in autonomous driving*. *IEEE Transactions on Intelligent Transportation Systems*, 23(10):17654–17665, 2022.

[23] Bing Yu, Haoteng Yin και Zhanxing Zhu. *Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting*. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, IJCAI-2018, σελίδα 3634–3640. International Joint Conferences on Artificial Intelligence Organization, 2018.

[24] U.S Department Of Transportation. Federal Highway Administratio. *Interstate 80 Freeway Dataset*, 2006.

[25] Nima Khairdoost, Mohsen Shirpour, Michael A Bauer και Steven S Beauchemin. *Real-time driver maneuver prediction using LSTM*. *IEEE Transactions on Intelligent Vehicles*, 5(4):714–724, 2020.

[26] SM Zabihi, Steven S Beauchemin και Michael A Bauer. *Real-time driving manoeuvre prediction using IO-HMM and driver cephalo-ocular behaviour*. *2017 IEEE Intelligent Vehicles Symposium (IV)*, σελίδες 875–880. IEEE, 2017.

[27] Nachiket Deo, Akshay Rangesh και Mohan M. Trivedi. *How Would Surround Vehicles Move? A Unified Framework for Maneuver Classification and Motion Prediction*. *IEEE Transactions on Intelligent Vehicles*, 3(2):129–140, 2018.

[28] Adam Houenou, Philippe Bonnifait, Véronique Cherfaoui και Wen Yao. *Vehicle trajectory prediction based on motion model and maneuver recognition*. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, σελίδες 4363–4369, 2013.

[29] SPYROS MAKRIDAKIS και MICHÈLE HIBON. *ARMA Models and the Box–Jenkins Methodology*. *Journal of Forecasting*, 16(3):147–163, 1997.

[30] Sima Siami-Namini και Akbar Siami Namin. *Forecasting Economics and Financial Time Series: ARIMA vs. LSTM*, 2018.

[31] Lirong Yao και Yazhuo Guan. *An Improved LSTM Structure for Natural Language Processing*. *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*, σελίδες 565–569, 2018.

[32] Developer Advocate IBM Anna Gutowska AI Engineer. *What is stochastic gradient descent?*, 2025. Accessed: 2025-10-20.

[33] Diederik P. Kingma και Jimmy Ba. *Adam: A Method for Stochastic Optimization*, 2017.

[34] Frank Rosenblatt. *The perceptron: a probabilistic model for information storage and organization in the brain.* *Psychological review*, 65(6):386, 1958.

[35] Sepp Hochreiter και Jürgen Schmidhuber. *Long Short-Term Memory*. *Neural Computation*, 9(8):1735–1780, 1997.

[36] Wai Cheong Tam, Eugene Yujun Fu, Richard Peacock, Paul Reneke, Jun Wang, Jiajia Li και Thomas Cleary. *Generating synthetic sensor data to facilitate machine learning paradigm for prediction of building fire hazard*. *Fire technology*, 59(6):3027–3048, 2023.

[37] Fernando Juan Pérez-Porras, Paula Triviño-Tarradas, Carmen Cima-Rodríguez, Jose EmilioMeroño de Larriva, Alfonso García-Ferrer και Francisco Javier Mesas-Carrascosa. *Machine learning methods and synthetic data generation to predict large wildfires*. *Sensors*, 21(11):3694, 2021.

[38] Amin Assadzadeh, Mehrdad Arashpour, Ioannis Brilakis, Tuan Ngo και Eirini Konstantinou. *Vision-based excavator pose estimation using synthetically generated datasets with domain randomization*. *Automation in Construction*, 134:104089, 2022.

[39] Moon Ye-Bin, Nam Hyeon-Woo, Wonseok Choi, Nayeong Kim, Suha Kwak και Tae Hyun Oh. *SYNAuG: Exploiting Synthetic Data for Data Imbalance Problems*, 2024.

[40] Ayesha Siddiqua Dina, AB Siddique και D Manivannan. *Effect of balancing data using synthetic data on the performance of machine learning classifiers for intrusion detection in computer networks*. *Ieee Access*, 10:96731–96747, 2022.

[41] Antonio J Rodriguez-Almeida, Himar Fabelo, Samuel Ortega, Alejandro Deniz, Francisco J Balea-Fernandez, Eduardo Quevedo, Cristina Soguero-Ruiz, Ana M Wägner και Gustavo M Callico. *Synthetic patient data generation and evaluation in disease prediction using small and imbalanced datasets*. *IEEE journal of biomedical and health informatics*, 27(6):2670–2680, 2022.

[42] Mauro Giuffrè και Dennis L Shung. *Harnessing the power of synthetic data in healthcare: innovation, application, and privacy*. *NPJ digital medicine*, 6(1):186, 2023.

[43] James Jordon, Jinsung Yoon και Mihaela Van Der Schaar. *PATE-GAN: Generating synthetic data with differential privacy guarantees*. *International conference on learning representations*, 2018.

[44] Fan Liu, Zhiyong Cheng, Huilin Chen, Yinwei Wei, Liqiang Nie και Mohan Kankanhalli. *Privacy-preserving synthetic data generation for recommendation systems*. *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, σελίδες 1379–1389, 2022.

[45] Cedric Deslandes Whitney και Justin Norman. *Real risks of fake data: Synthetic data, diversity-washing and consent circumvention*. *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, σελίδες 1733–1744, 2024.

[46] Krishnakant Singh, Thanush Navaratnam, Jannik Holmer, Simone Schaub-Meyer και Stefan Roth. *Is synthetic data all we need? benchmarking the robustness of models trained with synthetic images*. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, σελίδες 2505–2515, 2024.

[47] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez και Vladlen Koltun. *CARLA: An Open Urban Driving Simulator*. *Proceedings of the 1st Annual Conference on Robot Learning*Sergey Levine, Vincent Vanhoucke και Ken Goldberg, επιμελητές, τόμος 78 στο *Proceedings of Machine Learning Research*, σελίδες 1–16. PMLR, 2017.

[48] Weichao Qiu και Alan Yuille. *Unrealcv: Connecting computer vision to unreal engine*. *European conference on computer vision*, σελίδες 909–916. Springer, 2016.

[49] Shital Shah, Debadeepta Dey, Chris Lovett και Ashish Kapoor. *Airsim: High-fidelity visual and physical simulation for autonomous vehicles*. *Field and service robotics: Results of the 11th international conference*, σελίδες 621–635. Springer, 2017.

[50] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta και others. *Lgsvl simulator: A high fidelity simulator for autonomous driving*. *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*, σελίδες 1–6. IEEE, 2020.

[51] LG Silicon Valley Lab. *LGSVL Github Repo*, 2022. Accessed: 2025-10-20.

[52] Jean Emmanuel Deschaud. *KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator*. *arXiv preprint arXiv:2109.00892*, 2021.

[53] Andreas Geiger, Philip Lenz, Christoph Stiller και Raquel Urtasun. *Vision meets robotics: The kitti dataset*. *The international journal of robotics research*, 32(11):1231–1237, 2013.

[54] Tom Bu, Xinhe Zhang, Christoph Mertz και John M Dolan. *Carla simulated data for rare road object detection*. *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, σελίδες 2794–2801. IEEE, 2021.

[55] Yuhang Han, Zhengtao Liu, Shuo Sun, Dongen Li, Jiawei Sun, Chengran Yuan και Marcelo H Ang Jr. *Carla-loc: synthetic slam dataset with full-stack sensor setup in challenging weather and dynamic environments*. *arXiv preprint arXiv:2309.08909*, 2023.

[56] CARLA Simulator Docs. *get_forward_vector()*. Accessed: 2025-10-06.

[57] CARLA Simulator Docs. *Town 4*. Accessed: 2025-10-06.

[58] CARLA Simulator Docs. *Town 10*. Accessed: 2025-10-06.

List of Abbreviations

FSD	Full Self-Driving
V2V	Vehicle-to-Vehicle
LSTM	Long Short-Term Memory
GRU	Gated Reccurent Unit
RMS	Root Mean Square
CNN	Convolutional Neural Network
MLP	Multi-Layer Perceptron
RNN	Recurrent Neural Network
ML	Machine Learning
AI	Artificial Intelligence
ANN	Artificial Neural Network
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
MAE	Mean Absolut Error
mmap	Memory-Mapped
SGD	Stochastic Gradient Descent