TU WIEN Informatics

Multimodal RGB-D Autonomous Agents Steered by Deep Neural Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Technische Informatik

eingereicht von

Mihaela-Larisa Clement, BSc.
Matrikelnummer 12118529

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Dr.h.c. Radu Grosu
Mitwirkung: Mónika Farsang, MSc.

Wien, 26. März 2025

Mihaela-Larisa Clement	Radu Grosu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Multimodal RGB-D Autonomous Agents Steered by Deep Neural Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Computer Engineering

by

Mihaela-Larisa Clement, BSc.
Registration Number 12118529

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Dr.h.c. Radu Grosu
Assistance: Mónika Farsang, MSc.

Vienna, March 26, 2025

Mihaela-Larisa Clement	Radu Grosu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Mihaela-Larisa Clement, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 26. März 2025

Mihaela-Larisa Clement

Acknowledgements

Firstly, I am sincerely grateful to my advisor, Professor Radu Grosu, for the opportunity to work on this project, for his invaluable guidance and support throughout this project, and my studies at TU Wien.

Ultimately, I sincerely thank Mónika Farsang, whose hands-on supervision, patience, and dedication were essential at every stage of this thesis. Not only was your excellent advice during our meetings most valuable, but also your encouragement along the way has made all the difference in achieving the results.

I am also incredibly grateful to Felix Resch for his help with data collection, particularly for his expertise in synchronizing the collected data and setting up the data collection stack. Your contribution significantly helped me in the data preparation process and accelerated this project.

Notably, I am deeply thankful to the Scuderia Segfault autonomous racing team. The time I spent with you all has shaped me not just as a researcher, but as a more prepared and confident person.

Last but not least, thank you to my friends and family for your unconditional support and encouragement while pursuing this degree; especially to Mihai-Teodor Stănușoiu for also taking on the role of the human driver during data collection and testing sessions. Your involvement was crucial to the experiments, but more importantly, your unwavering confidence in me gave me the strength to persevere through every challenge in my studies.

Kurzfassung

Mit zunehmenden Fortschritten im Bereich des autonomen Fahren stellen genaue Entfernungsschätzungen und die Robustheit gegenüber Bildrauschen eine Herausforderung für ausschließlich auf RGB basierenden Steuergeräten dar. Diese Arbeit untersucht wie sich die Hinzunahme von räumlicher Tiefeninformationen, neben der Eingabe von "Single Modality", auf die Lenkvorhersage auswirkt. Im Gegensatz zu bestehenden Ergebnissen, welche die Vorzüge der multimodalen Wahrnehmung in Simulationen aufzeigen, werden in dieser Arbeit leichte RGB-D Modelle auf einer ressourcenbeschränkten Plattform eingesetzt.

Die Arbeit untersucht drei Schlüsselfragen: (1) *Inwieweit verbessert Tiefeninformationen die Lenkungsleistung?* (2) *Welche Fusionsmethode bietet die beste Balance zwischen Robustheit und Effizienz?* (3) *Kann volle Autonomie mit einem leichten multimodalen Steuerungssystem erreicht werden?* Zu diesem Zweck wird eine kleine autonome Plattform, der *roboracer*, verwendet, um verschiedene Fusionstechniken zu testen, einschließlich "early" und "late Fusion", sowie fortgeschrittene räumliche Anpassungsmechanismen wie "deformable convolutions" und Weitere. Neben Experimenten mit "open-loop" werden auch solche mit "closed-loop" durchgeführt. Die Leistung der Modelle wird anhand von Metriken wie der mittleren quadratischen Abweichung, "Attention Maps" und der Analyse von Steuerbefehlssequenzen mit Fokus auf der Navigationsstabilität bewertet.

Die Ergebnisse zeigen, dass die Einbeziehung von Tiefeninformationen durch "early Fusion" die Zuverlässigkeit des Controllers verbessert. Reine RGB-Modelle zeigen insbesondere bei unterschiedlichen Beleuchtungs Schwierigkeiten und versagen in Regionen mit unzureichenden visuellen Hinweisen, während RGB-D-Modelle Tiefeninformationen nutzen, um effektiv zu navigieren und selbst bei Bildausfällen und Sensorrauschen eine gleichbleibende Leistung aufweisen. Die "early Fusion" ermöglicht zudem eine robuste und stabile Navigation durch Fokussierung auf kritische Fahrhinweise. Dies bestätigt die Bedeutung einer tiefenverstärkten Modalitätswahrnehmung für robuste und effiziente Controller auf Basis neuronaler Netze.

Abstract

As more and more advancements are made in the autonomous driving scene, accurate distance estimation and image noise robustness remain challenging for solely RGB-reliant controllers. This thesis examines the impact of integrating depth information alongside single-modality input on steering prediction. Furthermore, stepping up from previous work which demonstrates the benefits of multimodal perception in simulations, this research deploys lightweight RGB-D models on a resource-constrained platform.

The presented work investigates three key questions: (1) *To what extent does depth improve steering performance?* (2) *Which fusion method provides the best balance between robustness and efficiency?* (3) *Can full autonomy be achieved with a lightweight multimodal control system?* A small-scale platform, *roboracer*, is used to test various fusion techniques, including early and late fusion, as well as advanced spatial adaptability mechanisms like deformable convolutions and variants thereof. Closed-loop experiments are carried out alongside open-loop counterparts while assessing the models' performance by employing metrics such as mean squared error, attention maps, and analysis of steering command sequences, with a focus on navigation stability.

Ultimately, the results reveal that incorporating depth information through early fusion improves the controller's reliability. In particular, RGB-only models struggle under varying lighting and fail in regions with insufficient visual cues, whereas RGB-D models leverage depth information to navigate effectively and maintain consistent performance even when exposed to frame drops and sensor noise. Above all, early fusion enables robust and stable navigation by focusing on critical driving cues, confirming the importance of depth-enhanced modality perception for robust and efficient neural network controllers.

Contents

CHAPTER 1

Introduction

Despite progress in autonomous driving, distance estimation and noise remain challenging when only RGB is used as input data. Prior research suggests that perception can be enhanced through RGB-D approaches, but these models have largely been confined to simulation. This gap is addressed in this work by asking:

1. To what extent does depth improve steering control?
2. Which fusion method provides the best balance between robustness and efficiency?
3. Moreover, can full autonomy be achieved with lightweight multimodal control?

Hence, this thesis tackles the deployment of steering agents that can prove to be robust and efficient in real-world conditions. This research demonstrates the real-world benefits of multimodal RGB-D perception in a small-scale autonomous system. Lightweight controllers that efficiently fuse RGB and depth are trained and assessed, achieving robustness while maintaining performance after deployment.

The thesis continues with Chapter 2, which aims to provide an introductory overview of the primary fundamental theoretical knowledge needed to understand the work. It starts with the simplest forms of neural networks, delving deeper until reaching deep learning models inspired by dynamical systems for continuous-time predictions.

In continuation, Chapter 3 analyzes previous solutions for similar autonomous agents. Recent research on neural networks for autonomous driving agents is reviewed, concentrating on three key areas: RGB-based approaches, multimodal fusion techniques, and state-of-the-art methods for the *roboracer* platform. Advancements that enable real-world deployment are emphasized, especially for resource-constrained platforms, as is the case for the *roboracer* platform.

The following chapter, Chapter 4, delves into the data collection process, as well as an overview of the *roboracer* platform used for autonomous driving research and its role in the thesis. The dataset generation process is explained, including the synchronization of RGB and depth data, preprocessing steps such as downscaling and interpolation, and final dataset labeling. The obtained dataset of synchronized RGB-D and steering commands is analyzed with respect to the driver's bias and the difficulty of the navigated tracks.

Chapter 5 provides both a high-level overview of the model architectures explored in this research, as well as the motivation behind the architectural choices made in the model networks. Among RGB-D fusion techniques, early and late fusion represent commonly used approached for multimodal perception. In addition, deformable convolutions (DCN and ZACN) were introduced, taking into account their possible spatial adaptability. Different RNN backbones were also evaluated to assess their sequential decision-making capabilities.

The results are presented in Chapter 6, separated into open-loop and closed-loop experiments. For the open-loop evaluation, Mean Squared Error (MSE) losses, as well as attention maps, were utilized for visual comparison. The chapter is structured into sections describing open-loop and closed-loop experiments. Open-loop evaluation focuses on performance metrics and visual analysis to compare different models, while closed-loop experiments assess real-world deployment on hardware. The behavior of the models is examined through attention maps and recorded trajectories, highlighting key strengths and failure cases.

Chapter 7 summarizes the main findings of this thesis and introduces potential directions for future research. The effectiveness of depth information for autonomous navigation is emphasized, with RGB-D models achieving full autonomy in a new scenario. Among fusion strategies, early fusion proved to be the most robust, maintaining stability even under sensor noise and hardware limitations.

CHAPTER 2

Background

Over the last decade, the machine-learning field has been rapidly evolving. Research is being conducted globally in a myriad of branches, driven by growing hardware capabilities and increased accessibility. Thus, we now have various architectural approaches to choose from and build upon. This calls for a solid understanding of the basis of our models in order to maximize their benefits.

With this in mind, this chapter introduces the **theoretical concepts** that form the foundation of the model architectures proposed in Chapter 4.

2.1 Deep Neural Networks Overview

2.1.1 Elementary Units of Neural Networks

As presented in [Zho21], the concept of **artificial neurons** was introduced in 1943 by Warren McCulloch and Walter Pitts in "A Logical Calculus of the ideas Imminent in Nervous Activity" [MP43]. The McCulloch–Pitts (MP) neuron model presented in Figure 2.1 is inspired by the structure of biological neurons, taking binary input signals and producing a binary output. Therefore, "firing" neurons that are in a state of 1 at time-step t, or "not firing" if the neuron state is 0.

In order to simulate this, the weighted sum of the inputs in Equation (2.1) is calculated using $w_i \in \{-1, 1\}, i \leq n$, where n is the number of inputs and the weight values replicate positive and negative synapses. The result is then checked against a threshold (θ), leading to an output of 1 if the threshold is exceeded and 0 otherwise. An important observation is that the MP neuron **has no truly learnable parameters**. This is caused by the weights of the binary inputs being static.

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right) \tag{2.1}$$

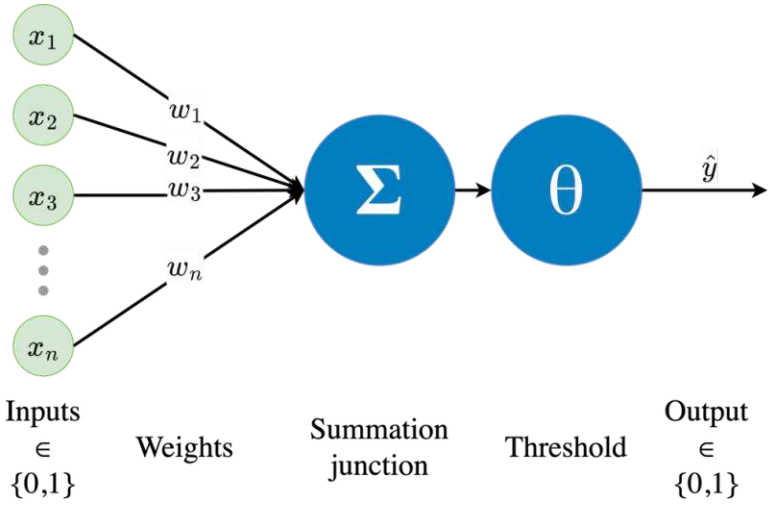


Figure 2.1: The McCulloch–Pitts neuron model.

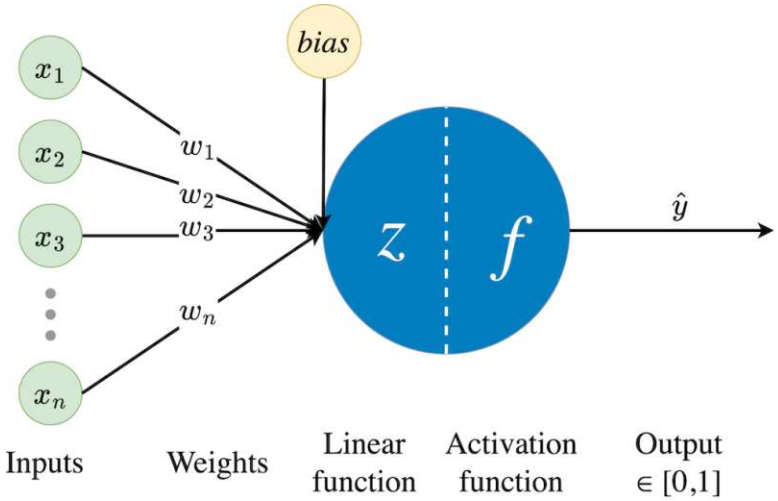


Figure 2.2: The Minsky perceptron model.

The Rosenblatt perceptron was introduced in 1958 in [Ros58] and can be presented as a refined artificial neuron. Firstly, it allows for real values as inputs as opposed to strictly $\{0, 1\}$ values. Secondly, assigning different values to each input opens the door to **learnable parameters**. Part of the learnable parameters is also the "bias" term. The **bias** can be seen as a learnable threshold θ by being associated with the weight of a static input with value 1. However, the output still remains split vertically by the θ value, making the Rosenblatt perceptron only suitable in binary classification scenarios.

In 1969, [MP69] (illustrated in Figure 2.2) improved the model by switching the Heaviside step function with a **sigmoid function** of Equation (2.2). This allowed the transition from binary outputs to real values in the $[0, 1]$ interval. As the sigmoid function is continuous

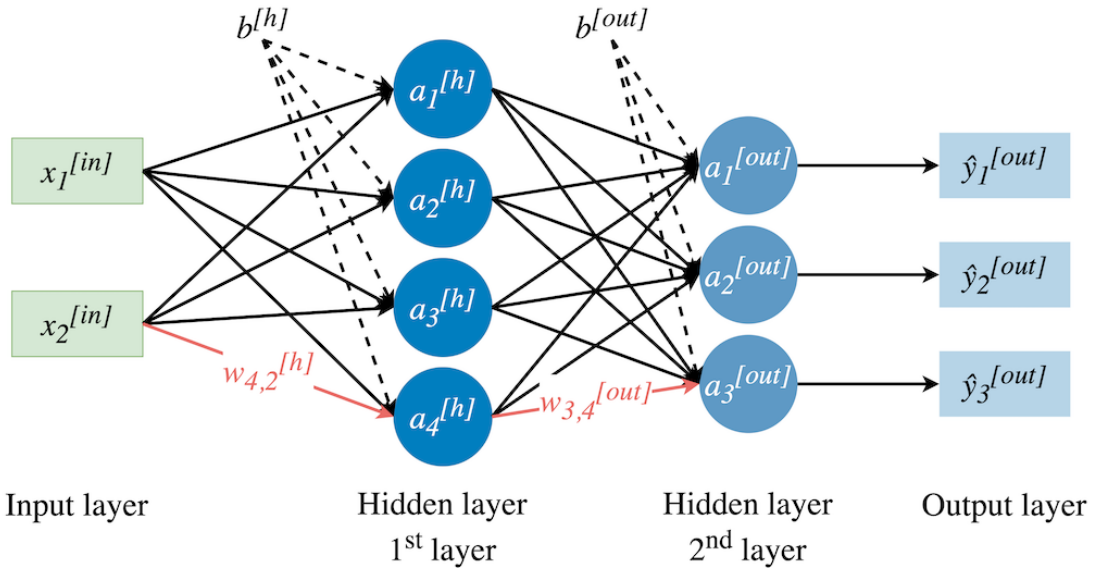


Figure 2.3: Example of an MLP architecture consisting of two hidden layers. At inference, data flows from left to right, whereas during backpropagation, it flows from right to left.

and differentiable, the Minsky perceptron can be used to tackle linear regression problems.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

2.1.2 Multi-layer Perceptron

Perceptrons constitute the **building blocks** for Multi-layer Perceptron (MLP)s, which represent elementary deep neural networks. As illustrated in Figure 2.3, they consist of several (at least 3) layers, each containing nodes (perceptrons or neurons). The first and last layers are commonly named the input and output layers, while all layers in between are called "hidden" layers.

The input to the network is given in the form of vectors, and as such, each neuron in the input layer represents a feature in the vector. For example, if the input is a greyscale image of 120x212 pixels, the input would be a vector of size $1 * 120 * 212 = 25\,440$.

In an MLP the layers are **feed-forward** and **fully connected**. The former refers to the fact that there are no cyclic operations within a layer, as the output of one neuron must be fed to the next layer. The latter term indicates that each neuron in one layer is connected to all neurons in the next layer. Therefore, information flows in one direction, from the input to the output. To assess how well a network performs, we must calculate the error in our prediction.

In regression problems, we typically apply a **loss function** that takes as input the predicted and expected output of our model. Common choices are the Mean Absolute

Error (MAE) / L1 Loss (2.3), the Mean Squared Error (MSE) / L2 Loss (2.4), and the Huber Loss (2.5). Although the Huber loss is robust against outliers due to reducing the impact of extreme values by combining MSE and MAE, it introduces yet another hyperparameter, δ, which determines the threshold at which the loss function transitions from quadratic (MSE-like) to linear (MAE-like).

$$\mathcal{L}_{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.3}$$

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.4}$$

$$\mathcal{L}_{Huber} = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & \text{for } |y_i - \hat{y}_i| \leq \delta, \\ \delta \cdot (|y_i - \hat{y}_i| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases} \tag{2.5}$$

where y_i represents the ground truth output, $\hat{y}_i$ is the model prediction, δ is a threshold parameter.

In order to train an MLP for a specific task (e.g., image classification), we make use of the **backpropagation algorithm** [RHW86] ("backward error propagation"). During backpropagation, we compute how much each parameter in the network contributes to the overall error, which allows us to make targeted tuning that improves model performance.

Backpropagation systematically calculates the gradient of the loss function with respect to each parameter by applying the **chain rule**. The algorithm thus determines the influence of each weight and bias on the overall error. Once gradients are computed, **gradient descent** [Rud16] updates the network's parameters in the direction that reduces loss.

The step size for these updates is controlled by the so-called **learning rate**. Choosing a good learning rate is in itself a challenge since we need to strike a balance between the network's convergence speed and stability. It is known that in order to converge to an optimal solution, backpropagation needs to be repeated across many training samples.

2.2 Convolutional Neural Networks

As seen in the previous section, MLPs process data sequentially in a fully connected fashion. Although this structure works well for tabular data or simple classification tasks, MLPs struggle with high-dimensional data such as images due to a large number of parameters and computational inefficiency.

As image processing represents an important part of this project, we must now turn our attention to popular techniques in the field. Classical computer vision relies on manually designed filters to extract features from images. As presented in [Sze22], techniques such as the **Sobel** filter for edge detection (illustrated in Figure 2.5), the **Laplacian** filter

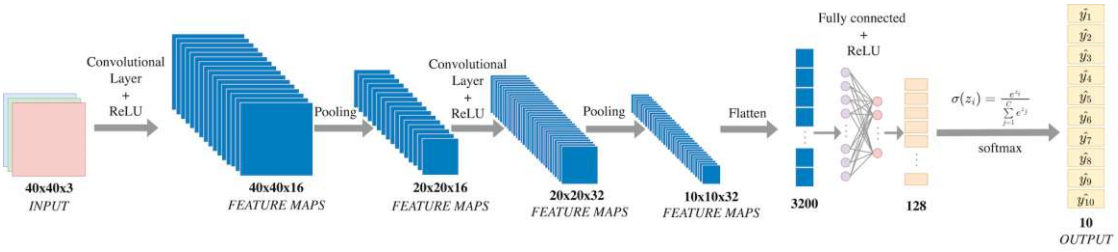


Figure 2.4: An example of a CNN architecture taking as input $40 \times 40 \times 3$ RGB images. The network consists of two convolutional layers with ReLU activation, each using a 3×3 kernel, stride 1, and 'same' padding. Each convolutional layer is followed by a 2×2 max pooling layer with stride 2, reducing the spatial dimensions. The output is flattened and passed through a fully connected layer with 128 neurons and ReLU activation, followed by a softmax layer for classification into 10 categories.

for detecting regions of rapid intensity change, and **Gabor** filters for texture analysis were widely used before the rise of deep learning. These hand-crafted feature extractors require significant experience and fine-tuning to perform well across different tasks.

Convolution is a *neighborhood operator* used to implement spatial filters and **feature detectors** in images similar to those previously described. The mathematical expression for this operation on an input grayscale image I is:

$$O(i,j) = \sum_{k=1}^{M} \sum_{l=1}^{N} I(i+k-1, j+l-1) K(k,l) \tag{2.6}$$

where K is a 2-D array called **"kernel"** or **filter**, O is the output pixel at (i,j) coordinates, and $i = \overline{1, M-m+1}$ and $j = \overline{1, N-n+1}$ [FPWW03].

A Convolutional Neural Network (CNN) [LBBH98] builds on this convolution operation from classical computer vision and has proven to learn optimal filters directly from data [HW18]. Instead of relying on predefined operations, CNNs iteratively adjust **filter weights** through backpropagation, enabling them to detect not only low-level patterns like edges but also complex, high-level representations such as shapes, textures, and even object parts. The need for manual feature extractors is thus reduced, enabling scalability to large datasets. From an architectural perspective, CNNs are feed-forward neural networks that use the convolution operation in their hidden **convolutional layers** (illustrated in Figure 2.4). Similarly to Equation (2.6), a kernel matrix is convolved with the input image in order to extract a different type of feature in each hidden layer. For example, one layer could learn to extract vertical edges, another one horizontal edges or blobs of color. As we move deeper into the network, these layers appear to learn increasingly complex representations [HW18], eventually detecting high-level objects and structures.

In terms of **hyperparameters**, we can tune the following ones to have a direct influence on the dimensions of the output coming from a convolutional layer:

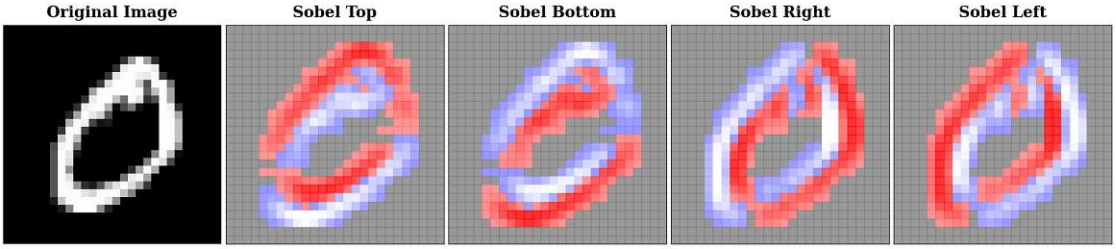


Figure 2.5: The figure displays the original MNIST digit '0' alongside the results of Sobel edge detection applied in four directions: top, bottom, right, and left. Each filtered image represents edge intensity, where red indicates positive activations, blue represents negative activations, while gray is used to depict neutral areas. A 3×3 Sobel kernel is convolved with the image with a stride of 1. Pixel intensities are used in a weighted sum with the kernel to pin regions of rapid intensity change. The highlighted edges correspond to the direction of each filter, revealing the structure and orientation of the digit's strokes.

- The number of **filters** determines the depth of the output feature map, with each filter learning a different feature.
- The **stride** defines how far the filter moves at each step. The size of the output feature maps decreases as the stride increases.
- The **padding** can be used to maintain spatial dimensions, typically by adding zero values around the input image (as 0 values have no effect on the operations).

Usually, after each convolutional layer, a **nonlinear activation function** is applied to introduce nonlinearity. This is typically the Rectified Linear Unit (ReLU) [KSH12]:

$$f(x) = \max(0, x) \tag{2.7}$$

ReLU ensures that negative values are replaced with zero, allowing only positive values to pass through. The derivative remains 1 for positive values, helping to mitigate the **vanishing gradient** problem.

To reduce the dimensions of the feature maps, CNNs can include **Pooling layers**. This facilitates retaining essential information while decreasing the number of parameters and, implicitly, the computational complexity. The most common pooling techniques include **max pooling** (selecting the maximum value from a defined window, thus preserving the most significant features) and **average pooling** (computing the average value of the window, "smoothening" the feature map).

Mathematically, max pooling can be expressed as:

$$O(i, j) = \max_{(m,n) \in R(i,j)} F(m, n) \tag{2.8}$$

where $O(i, j)$ is the output, $F(m, n)$ represents the feature map, and $R(i, j)$ is the pooling region.

Pooling can help improve translational invariance, i.e., small shifts or distortions in the input image do not drastically affect the output.

After multiple convolutional (and, optionally, pooling) layers, the extracted features are flattened into a one-dimensional vector and passed to a fully connected layer. This layer assigns class probabilities to the input image using an **activation function** like softmax:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{2.9}$$

where $P(y_i)$ represents the probability of class i, z_i is the input to the softmax function, and C is the total number of classes.

The fully connected layer enables CNNs to perform **classification tasks**, mapping extracted features to specific categories.

Over the years, various CNN architectures have been developed, each with innovations aimed at improving accuracy and computational efficiency. Some of the most influential architectures [IBM25] include: LeNet-5 [LBBH98], AlexNet [KSH12], VGGNet [SZ14], Inception [SLJ+15], ResNet [HZRS15], and DenseNet [HLVDMW17].

CNNs have transformed the field of image processing by efficiently extracting relevant visual features from inputs. While running at smaller computational costs than MLPs, CNNs combine convolutional, pooling, and fully connected layers while performing tasks. The rapid evolution of the research scene over the last decade has introduced deeper architectures and novel training techniques, which transformed CNNs into the backbone of modern computer vision applications.

2.2.1 Deformable Convolutional Networks

Deformable Convolutional Network (DCN) [DQX+17] enhances traditional CNNs by incorporating a mechanism to model geometric transformations, such as changes in object scale, pose, and deformation. Unlike standard CNNs, where the sampling grid for convolution (and pooling) operations is fixed, DCNs employ deformable convolution, which allows the grid to be dynamically adjusted through **learned offsets**, making the network more flexible in handling transformations.

The work replaces standard convolution with a new operation. In the new deformable convolution, the fixed grid locations are modified by additional offsets learned from the preceding feature maps. As presented in [DQX+17], the new operation can be expressed mathematically as follows:

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n). \tag{2.10}$$

where $y(p_0)$ is the output at position p_0, $\mathcal{R}$ is a fixed window that the filter will sample from in the input feature map, x is the input feature map, and w is the convolution weight.

Deformable convolution samples the input at **irregular locations**, aiming to provide the flexibility needed to adapt to different object scales and deformations. In [DQX+17], the offsets are computed by a convolutional layer applied to the input feature map (the same input that feeds into the regular convolution layer). This convolutional layer produces offset maps that specify how the regular grid should be deformed. Each offset is typically a small value that shifts the standard sampling positions either in the x or y direction (or both). These offsets are learned through backpropagation during training, similar to how convolutional weights are learned.

The offsets are fractional in nature [DQX+17], meaning they can represent non-integer shifts in the sampling locations. Since feature maps are discretized, these fractional positions are resolved by the authors using bilinear interpolation to compute the feature value at the non-integral location:

$$\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q}), \tag{2.11}$$

where G is the bilinear interpolation kernel.

The key difference from CNNs lies in the fact that deformable convolutions enable the model to learn how to "distort" its sampling grid based on the input, improving its ability to handle complex geometric transformations such as changes in object shape or viewpoint. As proven by the authors, the approach is particularly beneficial in tasks such as object detection and semantic segmentation, where objects can have significant variations in shape, scale, or pose.

Although DCNs are effective at adapting to geometric variations in object structures, their spatial support can sometimes extend beyond the relevant areas of an image, processing unnecessary content. To address this, the authors introduced DCNv2 [ZHLD18], which holds two new additions: the use of more deformable convolution layers and a modulation mechanism. They claim that these advancements enhance the network's ability to focus on relevant regions, using modulation scalar values between 0 and 1 to amplify or ignore features. Their experiments show that DCNv2 can outperform the original DCNv1 in object detection and instance segmentation tasks. However, modulation scalars need to be learned using new convolutional layers, which increase the size of the models.

2.2.2 Depth-Adaptive Offsets for Deformable Convolution

In standard deformable convolution, the sampling positions of convolutional kernels are adaptively shifted by learned offsets. As an alternative to learning the offsets via a second neural network, the authors of [WASD20] introduce ZACN. The key insight in their work is that offsets can be guided by a locally estimated **depth-adaptive plane** so as to capture geometric variations in the scene better.

Starting from a 3D point cloud representation of the scene, ZACN estimates a local depth-adaptive plane for every pixel, which helps approximate the surface at each pixel's location. The following equation governs such a plane:

$$n_1 x + n_2 y + n_3 z + d = 0, \tag{2.12}$$

where $\mathbf{n} = (n_1, n_2, n_3)$ is the normal vector of the plane (computed via Singular Value Decomposition (SVD) on the central pixel's neighbors within the receptive field in 3D space), and d is the plane offset.

Let R_{3D} be the obtained plane (3D sampling receptive field). The next step is to project it onto the image plane:

$$R' = \Pi(R_{3D}), \tag{2.13}$$

where $\Pi(\cdot)$ denotes the projection operation.

The final **ZACN offsets** are computed as the difference between the projected 2D sampling positions and the original regular grid positions:

$$\Delta p_n = R'(p) - R(p), \tag{2.14}$$

where $R(p)$ denotes the standard sampling position in the image domain, and $R'(p)$ is the adjusted sampling position obtained from the depth-adaptive projection.

In this way, ZACN offsets enable the convolutional kernel to distort in a manner that aligns better with the underlying 3D geometry of the captured scene.

2.2.3 Transposed Convolution

Deconvolutional Networks [ZKTF10] introduce a mathematical operation for synthesizing images from latent feature maps: transposed convolution. Unlike CNNs, which use a bottom-up approach, Deconvolutional Networks (DNs) adopt a top-down approach to **reconstruct** images.

DNs learn a hierarchical structure of features, where each layer builds upon the previous one, progressively capturing more complex representations. The reconstruction process is modeled as follows: given an input image y_i with K_0 color channels $y_i^1, ..., y_i^{K_0}$, each channel can be expressed as a sum of convolutions between learned filters $f_{k,c}$ and latent feature maps z_i^k [ZKTF10]:

$$y_i^c = \sum_{k=1}^{K_1} z_i^k * f_{k,c} \quad \forall\, c \in [1, K_0] \tag{2.15}$$

where $*$ denotes the convolution operation. However, since this system is underdetermined, a sparsity constraint is introduced in the form of a regularized cost function C_1 containing a sparsity-inducing norm $\|\cdot\|_p$.

Furthermore, to form a hierarchy, the latent feature maps $z_{i,k}^{l}$ at layer l serve as input to layer $l+1$, which leads to a more generalized cost function for higher layers C_l:

$$C_l(\mathbf{y}) = \frac{\lambda}{2} \sum_{i=1}^{I} \sum_{c=1}^{K_{l-1}} \left\| \sum_{k=1}^{K_l} g_{k,c}^{(l)} (z_{i,k}^{(l)} * f_{k,c}^{(l)}) - z_{i,c}^{(l-1)} \right\|_2^2 + \sum_{i=1}^{I} \sum_{k=1}^{K_l} \left\| z_{i,k}^{(l)} \right\|_p \tag{2.16}$$

where $g_{k,c}^{(l)}$ is a binary connectivity matrix that determines which feature maps from layer l connect to those in layer $l-1$.

DNs maintain sparsity while minimizing the error-prone image reconstruction process. Through the transposed convolution operation, each layer learns more abstract visual cues from the prior layer's features. Ultimately, the network learns to upsample feature maps and generate high-resolution image reconstructions.

2.2.4 Deformably-Scaled Transposed Convolution

Deformably-Scaled Transposed Convolution (DSTC) introduced in [BRX+22] is an upsampling method designed to improve feature reconstruction from DCNs. Unlike DN, which follows a fixed spatial relationship between input and output pixels, DSTC uses the **learned offsets** from a DCN to recover the original feature maps at each layer.

The transposed deformable convolution introduced in DSTC modifies the standard transposed convolution by introducing learnable offsets and interpolation kernels. Given the standard deformable convolution in Equation (2.10), the transposed deformable convolution modifies the receptive field **dynamically** by introducing the learned offsets in the transposed operation:

$$y(\mathbf{q}_n) = \sum_{\mathbf{p}_0} \mathbf{x}(\mathbf{p}_0) \cdot \mathbf{w}(\mathbf{p}_n) \cdot \mathcal{G}(\mathbf{q}_n, \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n) \tag{2.17}$$

where for each location $p_0 \in x$: $\Delta\mathbf{p}_n$ are the offsets learned, $q_n = r(p_0) + p_n + \Delta p_n$ are the offsets **locations**, $\{r(p_0) + p_n\}$ is the receptive field around $x(p_0)$, and G is a learned interpolation kernel for non-integer sampling locations.

One of the key motivations for DSTC is to better understand how features are learned and reconstructed in deformable convolution networks. Standard deformable convolutions adapt their receptive fields to capture spatially dynamic features. However, their impact on feature reconstruction during upsampling has been less explored. DSTC learns how to reconstruct those features effectively during decoding.

2.2.5 Visualization of CNNs for Steering Control

Originally, VisualBackProp [BCC+16] was developed for real-time debugging in autonomous driving applications. Essentially, it is a mechanism for obtaining attention maps, which highlight the most relevant regions of an input image that contribute to the

predictions of neural networks, in this case, CNNs. In contrast to other techniques, which rely on gradient computation (computing derivatives to determine the importance of each pixel), VisualBackProp takes a **value-based** approach. It builds on the intuition that as feature maps progress deeper into a CNN, they retain more important information, thus becoming more indicative of the final decision. By requiring fewer computations than a standard forward pass, VisualBackProp allows real-time visualization during both training and inference.

Arguably, the most important insight in [BCC+16] is to combine the advantages of both deep and shallow feature maps. Deep layers contain highly relevant but **low-resolution** information. Opposedly, shallow layers provide **finer spatial details** but may include less valuable features. VisualBackProp **backpropagates** information about relevant regions from deep layers while progressively refining the resolution using **deconvolution operations** (introduced in Subsection 2.2.3). The process results in a final attention map that emphasizes the most significant image regions that have contributed to a CNN's output. Since it does not rely on computing gradients, VisualBackProp offers a computational advantage over other visualization techniques like "Layer-wise Relevance Propagation" [BMB+16].

Another key insight of VisualBackProp is capturing sets of pixels working collaboratively rather than individual pixels in isolation. Unlike many visualization techniques that lack rigorous mathematical backing, VisualBackProp is grounded in a **network flow-based formulation**, allowing for a quantitative measure of how input pixels influence higher-level feature maps. Hence, it generates attention maps that highly reflect the network's decision-making process.

During evaluation [BCC+16], VisualBackProp has demonstrated its effectiveness in autonomous driving scenarios, where the generated attention maps highlight critical elements such as lane markings, vehicles, and road boundaries. Generally, the method is applicable to other domains where understanding CNN predictions is essential and provides a valuable tool for interpreting CNN-based systems.

2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are deep learning model architectures capable of processing **sequential data** by retaining **memory** of past inputs to influence future predictions. In contrast to traditional neural networks, which assume independence between inputs and outputs, RNNs maintain a **hidden state** that captures contextual information across time steps, given by Equation (2.18). The output of an RNN is produced by the Equation (2.19).

$$h_t = f(w_1 x_t + b_1 + w_2 h_{t-1} + b_2) \tag{2.18}$$

$$\hat{y} = g(w_3 h_t + b_3) \tag{2.19}$$

where x_t is the current input, $h_t - 1$ is the previous hidden state, w_1, w_2, w_3 are weight matrices, and b_1, b_2, b_3 are biases. The functions f, g are nonlinear (e.g., ReLU).

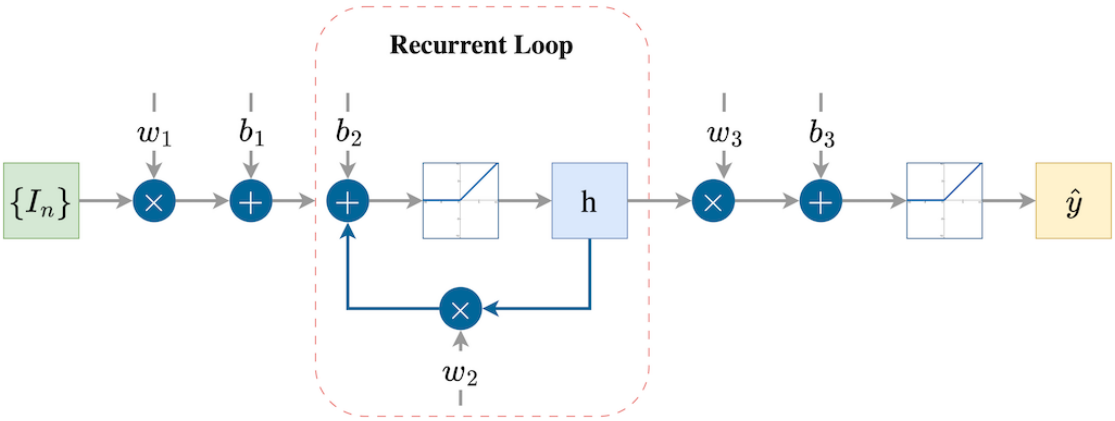


Figure 2.6: RNNs make predictions using sequences of input values. This is possible by means of the recurrent loop illustrated here in the basic model architecture. Unrolling an RNN at the recurrent loop would reveal n time steps for n inputs. The final output $\hat{y}$ is influenced by all inputs in the sequence. As the same weight w_2 is multiplied n times over the time steps, during backpropagation, the gradients can grow exponentially (when $w_2>1$) and vanish (when $w_2<1$).

However, standard RNNs struggle with long-term dependencies due to the vanishing gradient problem, depicted in Figure 2.6. Variants like Long Short-Term Memory (LSTM) address these limitations with specialized gating mechanisms that will be explained in the following section.

2.3.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks [HS97] are a specialized type of RNN designed to address the limitations of standard RNNs. As previously discussed, one of the major challenges with conventional RNNs is the vanishing or exploding gradient problem. In the case of the former, long-range dependencies become difficult to learn due to the exponential decay of gradient information over time. Meanwhile, in the latter issue, the weight updates can become excessively large, destabilizing training.

LSTMs consist of the following structural components (illustrated in Figure 2.7, based on [Yan16]):

- **Cell State** (c_t): A memory unit that carries useful information across time steps.
- **Input Gate** (i_t): Determines the extent to which new information is added to the memory.
- **Forget Gate** (f_t): Controls how much past information should be retained.
- **Output Gate** (o_t): Controls what information to use from the output derived from the memory cell.

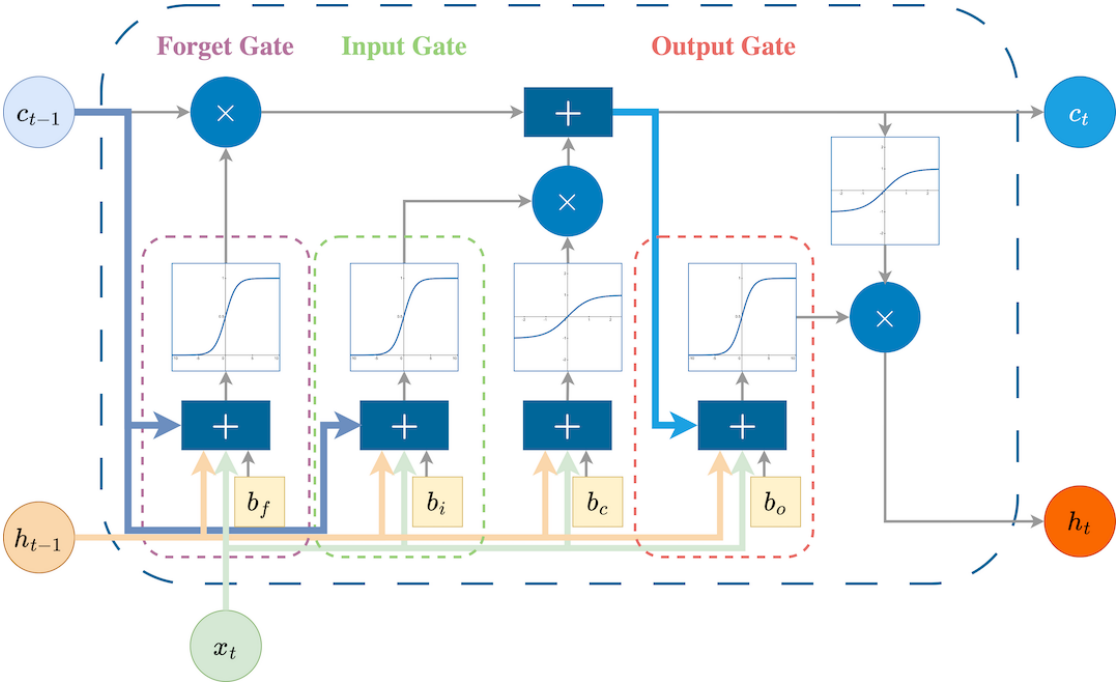


Figure 2.7: LSTM memory cell depicting the forget, input, and output gates. The thick colored arrows denote multiplications of the terms by the corresponding weight matrices in Equations (2.20), (2.21), (2.22), (2.23).

At each time step t, the LSTM updates its internal state using the following equations from [SSB14]:

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i) \tag{2.20}$$
$$f_t = \sigma(W_{fx}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.21}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \tag{2.22}$$
$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o) \tag{2.23}$$
$$h_t = o_t \odot \tanh(c_t) \tag{2.24}$$
$$y_t = W_{yh}h_t + b_y \tag{2.25}$$

where W denotes a weight matrix (e.g., W_{ix} is the weight matrix from the input gate to the input), b are bias terms, h is the cell output activation vector, σ is the sigmoid activation function, tanh is the hyperbolic tangent function, and $\odot$ denotes element-wise multiplication.

While simple RNNs update their hidden states directly, LSTMs maintain a separate *cell state* that allows information to persist across multiple time steps without being overwritten, regulated by the input and forget gate. This enables LSTMs to effectively learn long-term dependencies in sequential data, making them suitable for tasks such

as natural language processing, speech recognition, and time-series forecasting. By leveraging **gated mechanisms**, LSTMs avoid the instability and inefficiency seen in standard RNNs, ensuring stable gradient propagation and efficient learning over long sequences.

2.3.2 Neural ODEs

Two new insights in the direction of Neural ODEs were introduced in the paper of [CRBD18]. Firstly, a new class of continuous-time models, in which the hidden states are parameterized using an **ordinary differential equation (ODE) solver** instead of a discrete sequence of layers. Secondly, they introduced the **adjoint method** for backpropagation in ODE solvers, which computes gradients by solving an augmented ODE in reverse.

As opposed to RNNs, which require discretization of observations, ODE-Net represents each sequence as a latent trajectory governed by dynamic systems [CRBD18]. A dynamical system can generally be understood as a system that encompasses some state that changes with time (here, $h(t)$), along with a rule governing its evolution:

$$\frac{d\mathbf{h}(t)}{dt} = f(h(t), t, \theta) \tag{2.26}$$

where θ represents our system parameters.

Usually, this is a system of ordinary differential equations, for which the solution is obtained via integration, having been given an initial condition $h(t_0)$. In ODE-Net, f from Equation (2.26) is defined as a neural network, parameterized by the network weights, time becomes the depth of the model, and $h(t_0)$ is the input layer. To tune the accuracy of the solution obtained by ODE solvers, one can adjust the integration interval $\delta t = t_{next} - t$. In this way, the computations are adaptable to steeper or milder changes.

With respect to continuous-time backpropagation, the authors refer to the adjoint method. The following defines the forward and backward steps and the parameters:

$$a(t) = \frac{\partial L}{z(t)} \tag{2.27}$$

$$z(t+1) = z(t) + \int_t^{t+1} f(z(t))dt \tag{2.28}$$

$$a(t) = a(t+1) + \int_{t+1}^{t} a(t)\frac{\partial f(z(t))}{\partial z(t)}dt \tag{2.29}$$

$$\frac{\partial L}{\partial \theta} = \int_t^{t+1} a(t)\frac{\partial f(z(t), \theta)}{\partial \theta}dt \tag{2.30}$$

where $a(t)$ is the adjoint state, i.e., the gradient with respect to the hidden state at time-step t.

Through the use of continuous-time models, ODE-Nets improve long-term dependencies between inputs while avoiding exploding/vanishing gradients, which are common in basic RNN. Moreover, the methods employed in ODE-Net facilitate the modeling of dynamics in real-world time-dependent problems. This ensures a more robust handling of missing data.

2.3.3 Liquid Time-Constant Neural Networks

Liquid Time-Constant (LTC)s [HLA+20] are continuous-time recurrent models designed to improve expressivity and stability in time-series tasks. Instead of defining the system's evolution using fixed, implicit nonlinearities, LTCs employ first-order differential equations, where the time constants are dynamically adjusted via **nonlinear gating mechanisms**. This leads to a system where the rate of change of the hidden state adapts based on the input and the state, allowing for more flexible and robust sequence modeling.

LTC networks extend the standard ODE-Net formulation by introducing a **dynamic time constant**. In system dynamics, time constants arise from the idea that nothing happens instantaneously but with some delay. The time constant thus refers to the elapsed time during a system transition. The hidden state evolution, as stated in [HLA+20] is given by:

$$\frac{dx(t)}{dt} = -\left[\frac{1}{\tau} + f(x(t), I(t), t, \theta)\right] x(t) + f(x(t), I(t), t, \theta)A, \tag{2.31}$$

where τ represents a baseline time constant, $f(x(t), I(t), t, \theta)$ is a neural network with hidden state over time, and θ, A are parameters. This allows evolution at different rates of the hidden state.

One key advantage of LTCs is their *bounded dynamics*, ensuring stability even when inputs may vary significantly. The dynamic time-constant mechanism allows different neurons in the network to specialize in different timescales, improving the model's ability to capture complex temporal dependencies. Additionally, LTCs demonstrate higher *expressivity* compared to classical recurrent architectures, as measured by trajectory length in a latent space. Empirical evaluations on time-series prediction tasks confirm that LTCs outperform standard RNNs, including LSTMs and Neural ODEs, in several benchmarks. By leveraging a fused numerical ODE solver, the model maintains stability, making it a promising alternative for continuous-time sequence modeling.

2.3.4 Closed-form Continuous-time Neural Networks

Undoubtedly, continuous-time neural networks based on differential equations (DEs) have demonstrated strong capabilities in sequential decision-making. However, their efficiency is unarguably constrained by the computational load of the numerical solvers used to approximate these DEs. With this in mind, researchers have derived an **approximate closed-form solution** [HLA+22], which aims to solve the issue specifically for LTC

models. The authors eliminate the reliance on numerical DE solvers by computing a **tightly-bounded approximation** of the previously unsolved integration in LTC dynamics, leading to substantial speed improvements in both training and inference. In an attempt to avoid numerical solvers, the authors of [HLA+22] offer an approximate closed-form solution to the LTC from Equation (2.31). The key transformation in the Closed-form Continuous-time Neural Network (CfC) neural network model is:

$$x(t) = \sigma\left(-f(x, I; \theta_f)t\right) \odot g(x, I; \theta_g) + \left(1 - \sigma\left(-f(x, I; \theta_f)t\right)\right) \odot h(x, I; \theta_h) \tag{2.32}$$

where σ is a time-decaying sigmoidal function, the terms with $\sigma\left(-f(x, I; \theta_f)t\right)$ represent time-continuous gating, f, g and h are neural network instances.

This equation effectively replaces the ODE solver with an explicit time-dependent gating mechanism [HLA+22], making CfC significantly faster while maintaining expressive power.

CfCs offer a significant advantage over traditional ODE-based models by making time an explicit component of the formulation. As a result, the proposed models achieve speed-ups of one to five orders of magnitude [HLA+22] while maintaining the expressivity of DE-based networks. As such, CfCs appear to be an exciting alternative for resource-constrained platforms at inference time.

2.3.5 Liquid Resistance Liquid Capacitance Networks

As discussed in the previous section, LTCs often exhibit oscillatory behavior and require computationally expensive solvers. Liquid Resistance Liquid Capacitance Networks (LRC) [FNG24] extends LTCs by incorporating a *liquid capacitance*, which dynamically adapts the response of each neuron. While LTCs assume a fixed and constant membrane capacitance (1), LRCs introduce an additional *elastance* term (the reciprocal of capacitance) that depends on the neural state and the inputs. The equation [FNG24] for an LRC neuron is:

$$\dot{h}_i = \epsilon(w_i)\left(-\sigma(f_i)h_i + \tau(u_i)e_i\right), \tag{2.33}$$

where h_i is the hidden state of neuron i, $\epsilon(w_i)$ is the **elastance**, an adaptive function that modulates capacitance, $\sigma(f_i)$ represents a forget gate, $\tau(u_i)$ is an update gate, e_i is the resting potential.

In LTCs, neurons react to inputs with a fixed responsiveness due to the capacitance which is assumed to be constant. LRCs instead introduce $\epsilon(w_i)$, which dynamically scales the effect of input signals and hidden states, **reducing unnecessary variations**. This modification allows LRCs to dampen oscillations while maintaining high expressivity.

A key benefit of LRCs is that their smoother dynamics enable efficient numerical integration. While LTCs often require high-order solvers to prevent instability, LRCs can be solved accurately with a simple explicit Euler method using just one unfolding step.

This leads to a highly efficient recurrent unit, called the **LRC unit (LRCU)**, which functions as a biologically inspired alternative to GRUs and LSTMs.

All in all, LRCs refine the modeling of continuous-time recurrent dynamics by introducing an adaptive capacitance, which improves numerical stability and generalization. By reducing oscillations while preserving expressive power, LRCs serve as an efficient and interpretable alternative to both Neural ODEs and traditional gated RNNs.

CHAPTER 3

Related Work

This chapter gives an overview of the recent studies in neural networks employed for autonomous driving agents. The chapter consists of RGB-based approaches, the fusion of different modalities, and state-of-the-art methods for the *roboracer* platform. Emphasis is made on crucial advancements, real-world deployment and solutions for resource-constrained platforms.

Overall, the following studies emphasize the shift in autonomous systems toward multimodal learning and compact neural controllers, yet real-world deployment remains limited. This thesis aims to enable autonomous agents to navigate their environment reliably on low-cost, resource-constrained hardware. By deploying efficient RGB-D-based models, the practicality and robustness of models can be brought over to real-world scenarios.

3.1 Unimodal RGB Input

First and foremost, one key achievement in the end-to-end autonomous driving setting is [BTD+16]. The authors trained a 9-layer CNN on 72 hours of RGB input, which made successful steering predictions actuating a drive-by-wire vehicle. Even though the visualization of learned features indicated the model had learned valuable road features, attempts to assess which areas of the network contributed to decision-making were not achievable. This is due to the constructed network comprising of 27 million connections and 250 thousand parameters, which hindered interpretation tremendously.

The recent research of [LHA+20] introduced a compact bio-inspired neural controller, called NCP, which facilitated interpretability during real-time deployment, an issue that traditional deep learning methods pose. While using a 79,420 parameter CNN for feature extraction, they utilize as controller a hierarchically structured LTC which focuses on short-term causal relationships, resulting in a 80,485 parameter model. Their proposed

NCP, consisting of only 19 neurons and 253 synapses, significantly out-performed purely CNN methods both in simulation and hardware deployment using RGB data.

In a different approach, [ZGL+20] aimed to overcome the limitations of traditional CNN-based steering models by introducing a temporal dependency-aware model. A C-LSTM network architecture is introduced with a 4-layer CNN for feature extraction, followed by 2 LSTM layers and 3 FC layers, amounting to a total of 800 thousand parameters. After training with more than 40 thousand RGB images obtained from a simulated environment, the dataset evaluation of the model proved effective feature extraction and good autonomy.

Lastly, [MBB+24] makes use of optical flow from RGB frames to predict steering angles. The authors leverage CNN-based feature extraction with RNN's, mainly LSTM and NCP. They assessed both early and hybrid fusion techniques and demonstrated that incorporating optical flow significantly reduces steering estimation error — by 31% compared to RGB-only methods, thus highlighting the potential of leveraging motion dynamics for enhanced vehicle control.

3.2 Multimodal Input

In this section, several studies are presented, all employing RGB-D information with the scope of better autonomous control.

In particular, [MVLÁCBAM24] introduces a Recurrent CNN (RCNN) trained with RGB-D images, where the depth data is obtained through the fusion of a camera and a LiDAR sensor in a simulated ROS environment [MVLÁCBAM24]. This study emphasizes the importance of multimodal perception for enhancing vehicle autonomy, demonstrating that the proposed neural network achieves an autonomy rate of 95.9%, a 9.95% improvement over its base form.

Likewise, another study [XCG+19] delves into the effect of multimodal perception for network-controlled cars, with a focus on early, mid, and late fusion techniques. The authors use the CARLA simulator and a branched Conditional Imitation Learning (CIL) model to train and test models using single-modality (RGB or depth) and multimodal (RGB-D) approaches. They reached the conclusion that early fusion multimodality significantly outperforms unimodality, especially under challenging driving conditions, with a success rate improvement from 46.67% (RGB) to 94.00% (RGB-D).

Both works highlight the advantages of incorporating depth information into autonomous driving models. While the first study focuses on improving RCNN architectures and evaluating their impact on vehicle autonomy, the second study provides a comprehensive analysis of different fusion techniques in an end-to-end learning paradigm. These findings reinforce the growing consensus that multimodal sensor fusion, particularly RGB-D integration, enhances perception and decision-making in autonomous driving systems.

Beyond traditional vision-based methods, the role of event cameras in autonomous navigation has also been explored. A recent study introduced DRFuser, a multimodal fusion

network that combines RGB and event-based vision using a self-attention mechanism within an encoder-decoder architecture [MALJ22]. By leveraging self-attention layers, the model effectively integrates spatial and temporal dependencies across both modalities, achieving good performance in dataset evaluation.

3.3 State-of-the-Art of *roboracer*

End-to-End (E2E) deep learning approaches have been widely explored for autonomous racing on the *roboracer* platform, with multiple works leveraging DRL for control [EBZ+23], [EJE23], while another study provides a comparative analysis of various DRL methods in autonomous racing [BDEE23].

In contrast, supervised learning-based approaches for *roboracer* are relatively unexplored, with TinyLidarNet [ZWY+24] being a key recent contribution. TinyLidarNet introduces a lightweight LiDAR-based 1-D convolutional model that achieves competitive racing performance while maintaining inference efficiency on low-end microcontrollers.

However, its reliance on LiDAR raises concerns about robustness in challenging environments. A separate experimental study has demonstrated that depth cameras significantly outperform LiDAR under ill-reflective conditions, maintaining full ranging capabilities where LiDAR degrades to just 33% of its nominal performance [LBG+23].

Given these findings, this thesis diverges from TinyLidarNet by adopting an RGB-D-based supervised learning approach, leveraging the enhanced robustness of depth sensing to improve autonomous performance.

Lastly, as it is well-known that IR cameras are less affected by ambient lighting conditions compared to RGB cameras, they represent a solid choice for this project's hardware application. Their reliability in varied lighting environments, including low light or bright sunlight, where visible light sensors might struggle, should help when the model is deployed in the real world, overcoming issues such as time of day differences between training datasets and active testing sessions.

CHAPTER 4

Setup

The platform that was used is a small-scale vehicle, known as *roboracer*. Univ. Prof. Radu Grosu introduced the platform to TU Wien as part of the "Autonomous Racing Cars" lecture.

Inspired by the lecture, students formed a *roboracer* autonomous racing team, Scuderia Segfault [Seg21]. The team has since then secured podium places repeatedly in international head-to-head racing competitions. The hardware used for this thesis was provided by the racing team, which operates as an extension of the Cyber-Physical Systems Research Unit.

4.1 Data Collection

The following section aims to give an overview of the hardware platform, as well as the process of data collection. The choices made at each step are explained, with details on the configurations used.

4.1.1 Hardware and Software Stacks

The platform contains a lower-level and an upper-level chassis [CFRG25], as well as hardware parts to help in providing autonomous support behavior. The vehicle instance used in the experiments is illustrated in Figure 4.1. The *roboracer* developers provide more detail about the system in [BZJ+24] and recent applications in academia [CFRG25].

The brushless electric motor resides on the lower-level chassis of the *roboracer* and is powered by a 3-cell LiPo battery. Meanwhile, the upper-level chassis holds the power board, a Vedder Electronic Speed Controller (VESC) for directing input signals to the motor and to the autonomy elements. The autonomy-enabling elements are a processing unit (13th Gen Intel(R) Core(TM) i7-1360P), LiDAR sensor, and Inertial Measurement Unit (IMU).

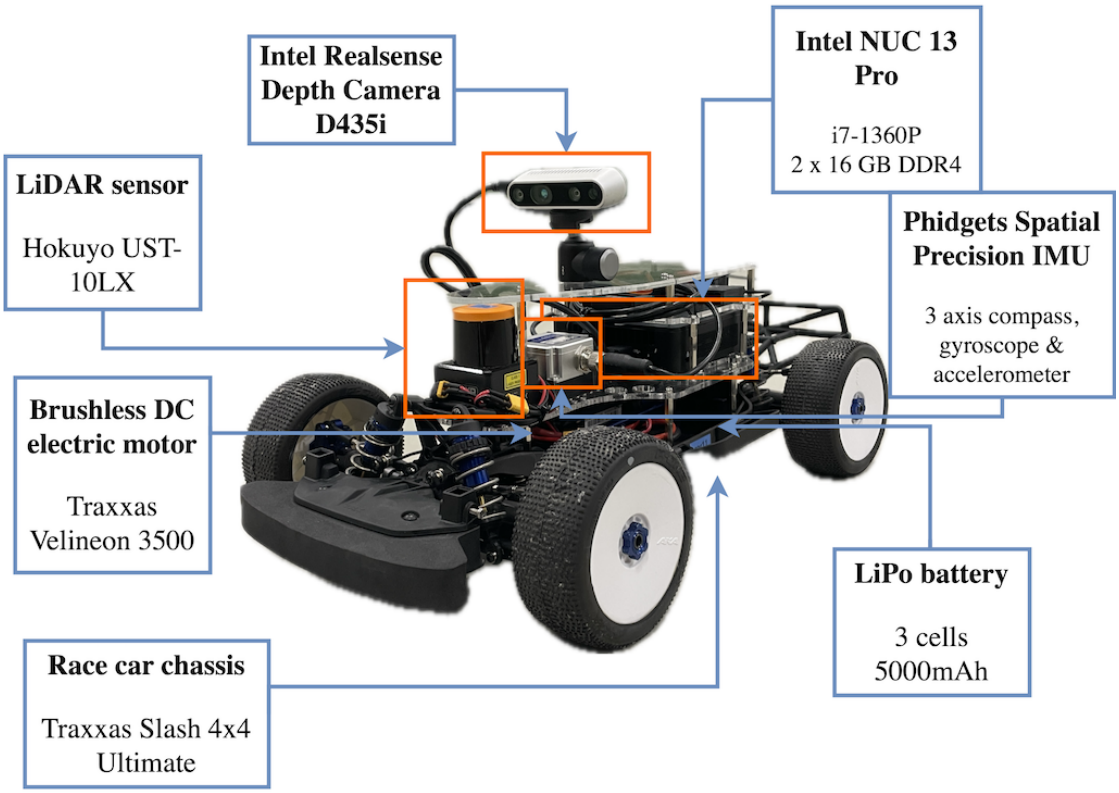


Figure 4.1: Component breakdown of the *roboracer* vehicle used for the experiments.

The unit on this *roboracer* instance uses only the CPU without dedicated hardware acceleration for machine learning inference. Therefore, all autonomy-related computations, including sensor fusion and decision-making, are executed solely on the CPU. Since there is no hardware acceleration, this platform is considered resource-constrained for deploying ML models; as a result, larger, more computationally demanding models may not deliver the required real-time performance.

Control-wise, the platform currently used during autonomous driving competitions by the racing team does not use optical lens vision. Therefore, a stereoscopic depth camera (Intel Realsense D435i, Figure 4.1) was introduced into the hardware setup [CFRG25].

The camera's datasheet gives the following information that is relevant to this thesis [CFRG25]:

- A detection distance of 10m, optimal in between 0.3m to 3m
- Field Of View (Horizontal×Vertical): RGB 69°× 42°, Depth 87°× 58°
- Depth RMS Error $\leq 2\%$, i.e $\approx 40mm$ RMSE at $\approx 2m$

Since the RGB camera settings allow only *30fps*, the depth setting was configured to the same number. With respect to the resolution, both streams were set to 848 × 480px, listed as the best option for the depth stream [CFRG25].

roboracer uses Robot Operating System (ROS) as its core software framework. ROS [Rob21] is an open-source middleware designed to facilitate the development and integration of complex robotic systems by organizing functionality into modular components known as nodes. The inter-node communication is done via channels (topics) which enable the sharing and coordination of data (messages) across the system. The ROS architecture is flexible and scalable, which facilitates the development of applications from sensor input to high-level planning.

The University of Pennsylvania provides enthusiasts with a base stack [Fou25], which provides the fundamental tools and libraries for processing sensor data, managing state, and interfacing with hardware components. As the Scuderia Segfault in-house stack builds on this foundation with more implementations, it was used for actuating the car in this project. As the complete stack deals with the core functionalities necessary for actuation, a new node was introduced dedicated to the closed-loop experiments. The node (written in Python) is subscribed to the RGB-D topic and publishes steering and constant speed commands on the */drive* topic. In an experiment instance for this project, the pre-trained model is loaded when the node is started and is used in inference mode to make predictions in the camera messages callback. The camera messages are converted to OpenCV images using *cv_bridge* [PMG24], scaled down to 212 × 120, and used by the model. Importantly, Intel Extension for PyTorch [Int25] is used in imperative mode to optimize the model performance.

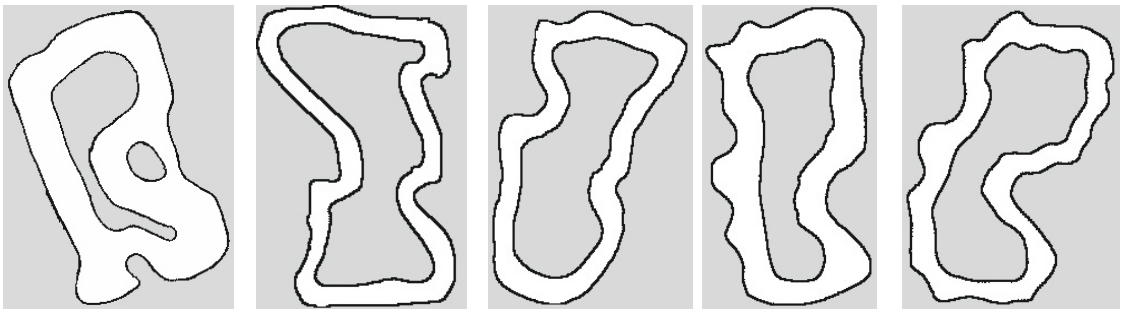
It is worth noting that there is an ongoing debate about the real-time capabilities of ROS. Critics argue that ROS is not truly real-time because it operates on a Linux environment, which is inherently best-effort rather than deterministic. Factors such as network communication delays, internal memory management, and the potential for non-deterministic scheduling mean that ROS is not able to meet the timing requirements of high-frequency control. However, the system can still achieve the performance necessary for robust operation by offloading critical timing-sensitive tasks to, for example, specialized threads.

4.1.2 Expert-Driven Data Acquisition

It is important to first describe the environment used for data collection. The driving area was set up indoors, employing pipes to build different circuits for driving. This environment was chosen because these pipes are commonly used during *roboracer* competitions, making them ideal for leveraging multimodal visual cues [CFRG25].

Moreover, it is notable that no attempts were made to control lighting conditions. In this way, recording sessions were carried out at different times during the day, with and without artificial lighting.

In total, a series of five circuits to obtain variations in turns, differing in curvature and segments, which are illustrated in Figure 4.2. Driving on all circuits was recorded in clockwise, as well as anti-clockwise direction for 6 minutes each time (4 minutes for Figure 4.2a, due to time restrictions) [CFRG25].



(a) (b) (c) (d) (e)

Figure 4.2: Circuits used during expert driving data recordings. From [CFRG25].

In what follows, the human control pipeline is described. A driver with 10 years of frequent driving experience was chosen to control the car on the different track configurations. The remote control of the car was carried out by a driver via steering wheel and pedals [CFRG25].

After multiple test runs on the track in Figure 4.2a, it became necessary to introduce a speed cap. It was found that because of the limited Field of View (FOV) of the camera, the driver was unable to drive with speeds more than 0.9m/s [CFRG25].

The data recording pipeline is fully illustrated in Figure 4.3. To create a first-person perspective, the onboard camera's stream was transmitted over the network to a secondary device displaying the footage on a monitor [CFRG25].

During recordings, all rostopic messages were saved to rosbags on the car, except the camera RGB topic. Since the RGB information was being redirected to the support device connected to the monitor, it had to be saved separately on that device [CFRG25].

Table 4.1 presents an overview of the essential properties of all the maps used in the data collection step. It must be noted that the driver was asked to drive test runs before the recording in order to ensure that each track could be navigated successfully [CFRG25]. A second note is that for Figure 4.2a, the driver was instructed to take the longer path when reaching the island [CFRG25].

Based on the small lap times of $\approx 33s$, it can be said that maps 4.2a and 4.2d were most effortless to navigate, regardless of driving direction. This result can be explained by the fact that both maps have the most extended straight sections, smooth, long turns, and long distances to the walls. Figure 4.2b was the hardest to navigate due to its narrowness, the rapid succession of turns, and sharp corners.

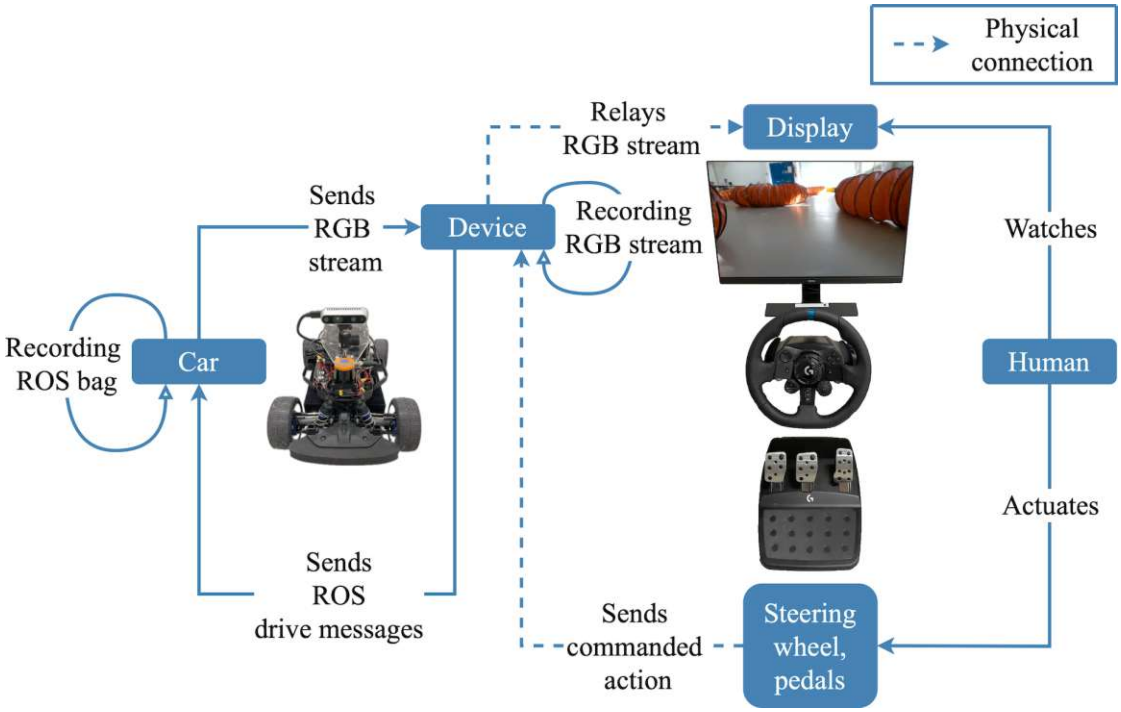


Figure 4.3: Diagram showing the vehicle control architecture used during recordings. From [CFRG25].

Map	No. Laps (CW/ACW)	Lap Time [s] (CW/ACW)	Length [m]	Width [m] avg. left/right
4.2a	7/6	33.30/33.80	33.52	0.74/0.91
4.2b	9/9	38.50/39.74	35.98	0.4/0.3
4.2c	9/9	36.04/37.53	32.17	0.49/0.41
4.2d	10/10	33.50/33.85	31.21	0.69/0.61
4.2e	9/9	38.06/37.54	33.72	0.49/0.42

Table 4.1: Quantifiers for assessment of the tracks driven on by human expert.

4.2 Dataset Generation and Preprocessing

The synchronization of the rosbags with the video recordings was done on a separate machine and was possible by using the common *30fps* [CFRG25]. By this, it is meant that while the messages in the rostopic for the depth camera were being recorded at *30fps*, the same frequency was used for the RGB stream.

Initially, as described in [CFRG25], the depth topic messages were retrieved from each rosbag [Res23], which contain *uint16* 1-D images and contain pixel-wise depth measurements. The frames were used to obtain colormaps, using each measurement with respect

to a constant value. After time-stamping the frames, colorful depth videos were generated for each track recording session.

In the next phase, manual synchronization was performed between the original videos and the depth data [CFRG25]. In preparation, the RGB videos were also time-stamped. For stream synchronization, signs were positioned in front of the car at the start of the recordings. Frame-wise stamps were manually registered for the identified common start and end of the video streams, including laps breakdown.

In continuation, the streams were resized to 212x120px. After trying different techniques (nearest neighbor, Gaussian, Lanczos [Lan50], Catmull-Rom [CR74]), Lanczos-3 (Figure 4.4d) was considered to be the best, with nearest neighbor (Figure 4.4a) giving poor results, the Gaussian counterpart (Figure 4.4b) blurry ones, and Catmull-Rom (Figure 4.4c) duller images [CFRG25].

Finally, to extract the original rosbag timestamp of the messages, the depth topic was parsed according to the logged recording segments. Using the value interpolation technique in [Res23], the pairs of drive messages and visual stream were identified and saved in npz format [CFRG25].

4.2.1 Dataset Analysis

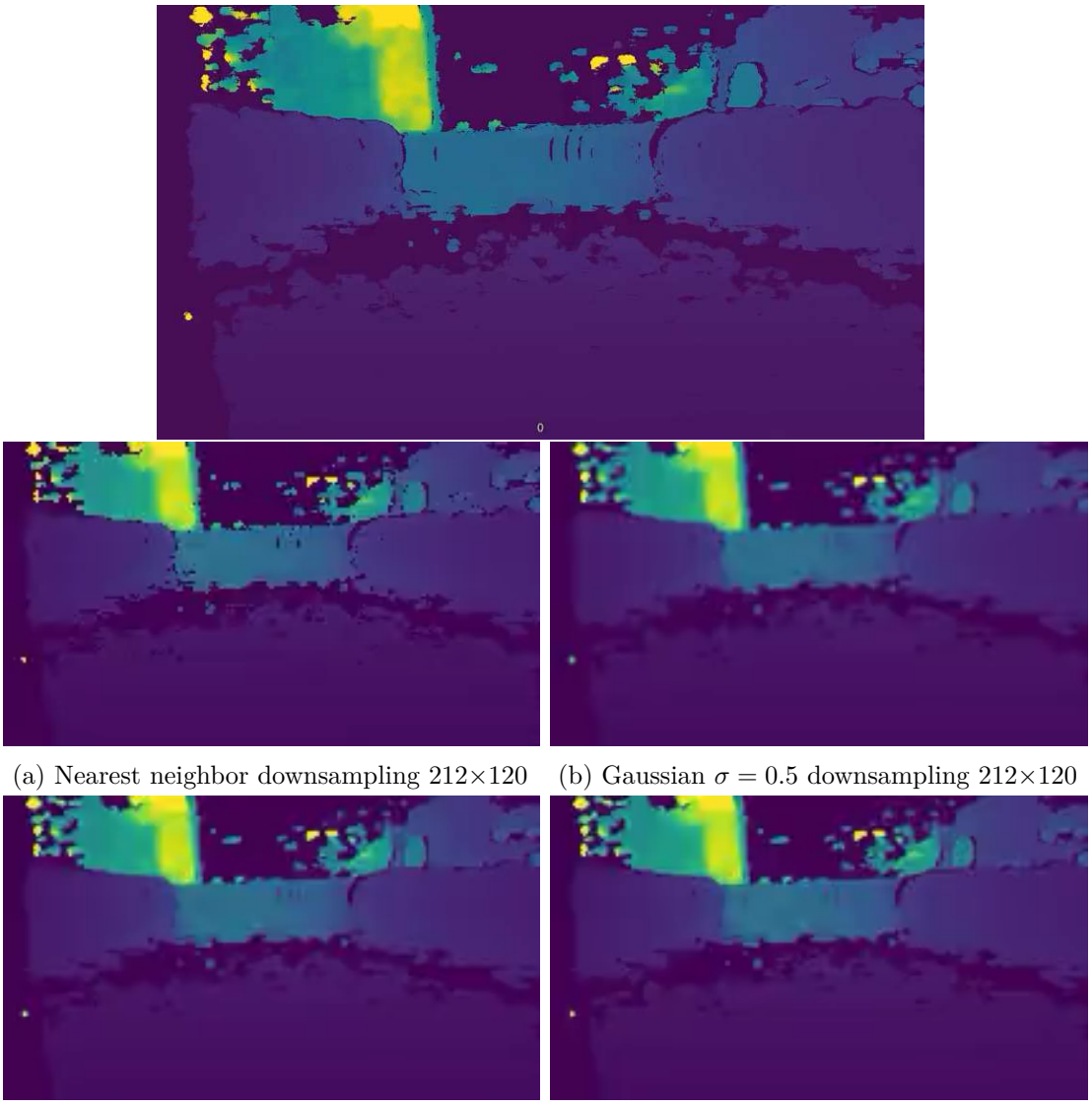
Thus, datasets totalling to $\approx 100\,000$ entries were generated [CFRG25]. Figure 4.5 offers an insight into not only the variety of commands reached on driven tracks but also the difference in distribution between driving directions.

Naturally, **anti-clockwise** driving directions keep strictly positive median values, which correspond to the driver keeping left while driving. In contrast, **clockwise** driving is characterized by negative median values.

Moreover, the spread of the distributions matched the levels of navigation difficulty previously implied: narrow distributions (tracks 1 and 4) were also faster to navigate, whereas the broadest distribution (track 2) resulted in the longest navigation times.

Lastly, the outliers on the Anti-clockwise (ACW) direction on tracks 2, 3, and 5 correspond to very sharp right turns.

Figures B.1 and B.2 show individual distributions generated from 1 lap on each track in anti-clockwise and clockwise directions. For ease of interpretation, bins have been normalized such that each bar sums up to 100, and smoothed Kernel Density Estimation (KDE) curves have been included. The normalization ensures proper representation, given that not all laps have the same number of observations. In this way, it is straightforward to conclude that the gathered dataset covers a wide range of steering commands.



(a) Nearest neighbor downsampling 212×120 (b) Gaussian $\sigma = 0.5$ downsampling 212×120

(c) Catmull-Rom bicubic downsampling 212×120 (d) Lanczos $a = 3$ downsampling 212×120

Figure 4.4: Colorized visualization of downsampled depth streams.

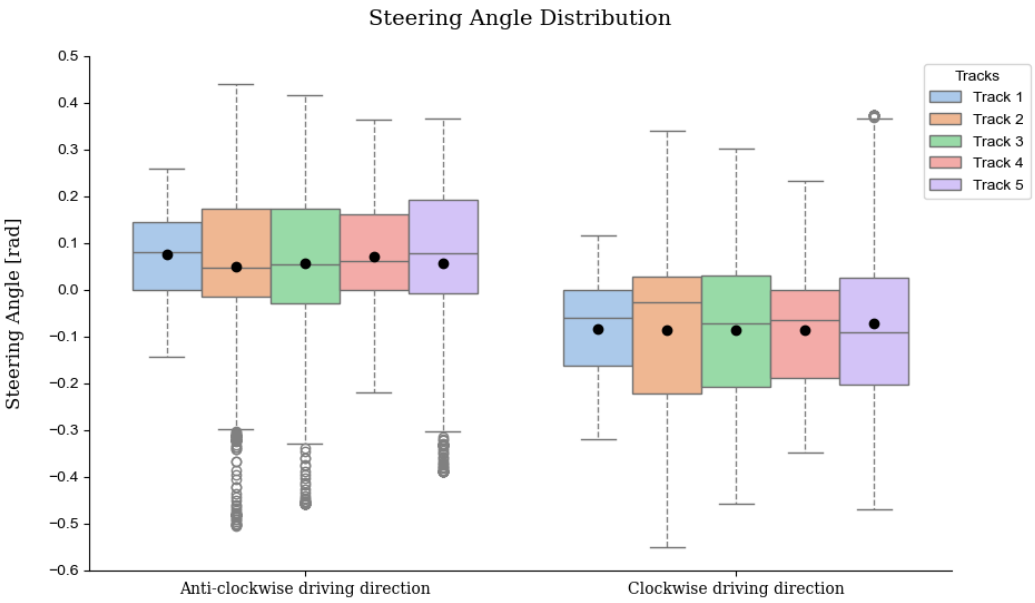


Figure 4.5: Distributions of recorded steering angles in 1 lap on each track. Negative values correspond to steering right, whereas positive values correspond to steering left.

CHAPTER 5

Implementation

The following sections provide a walk-through of the steps taken prior to and during training. These include setting up the environment, structuring the model architectures, and the choices made along the way. The modular software design that was put in place to implement reusable and maintainable code is described. In particular, the separate modules created for model instantiation, data loading, training routines, and evaluation are presented.

5.1 Work Environment

To train the models, a virtual machine with Ubuntu 20.04.6, 3 072 CUDA cores and 12 GB of GDDR5 memory was utilized [CFRG25]. Given the widespread use of PyTorch in computer vision and offered implementations, it was chosen for implementing the models, running in a Conda environment.

Several measures can be taken to ensure reproducibility, although identical behavior cannot be guaranteed between all different runs. With this in mind, all sources of non-determinism that are adjustable through the PyTorch framework are seeded with a chosen value on program start.

A dedicated data loading pipeline was implemented with seeded shuffling and handling of different file formats and image sizes, ensuring that the experimental setup can be easily adapted to future datasets and requirements. Using this dataset loader, the recordings were shuffled. The dataset was split into 60% training, 20% validation, and 20% test data, and the dataset for each recording containing laps is then further shuffled prior to use by models [CFRG25].

5.2 Model Architecture

5.2.1 High-level Overview

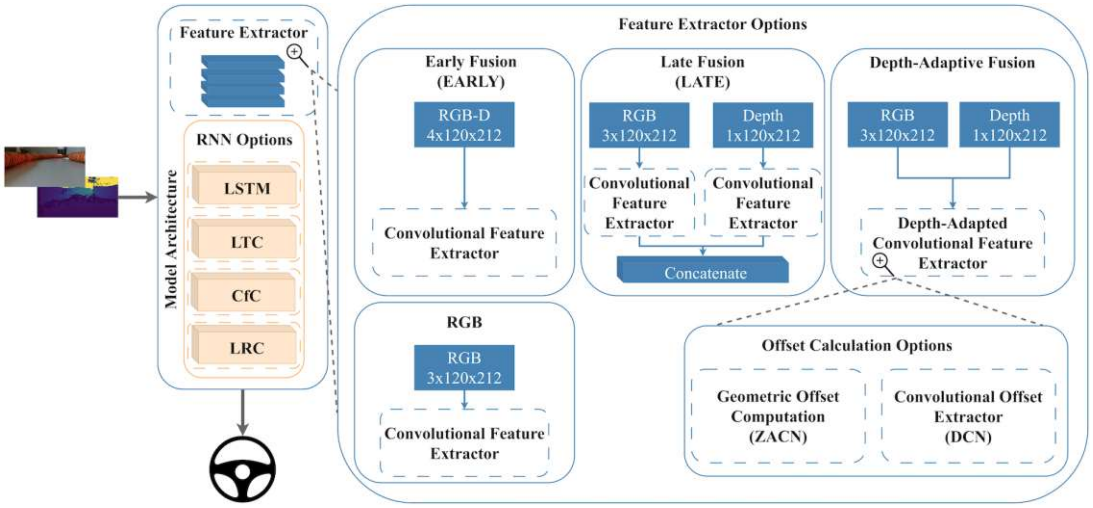


Figure 5.1: Structural diagram of the network architectures used in this thesis, in a channels-first format. From [CFRG25].

Starting from the approach of the authors of [LHA+20], twenty architectures were designed and implemented, published in [CFRG25], as explained in Appendix A:

- Four of the architectures diverge from [LHA+20] based on the integration of multimodal input and different approaches to achieving this
- One architecture emulates the original method
- There are four Recurrent Neural Network (RNN) options: Liquid Time-Constant (LTC), Closed-form Continuous-time Neural Network (CfC), Long Short-Term Memory (LSTM), Liquid Resistance Liquid Capacitance Networks (LRC)

The common sequence length for all RNNs is sixteen. In Closed-form Continuous-time Neural Network (CfC) and Liquid Time-Constant (LTC), an Neural Circuit Policy (NCP) with 19 neurons (12 inter-neurons, 6 command neurons, 1 motor neuron) was used [CFRG25]. Long Short-Term Memory (LSTM) and LRC employ 64 features in their hidden state [CFRG25].

5.2.2 Modular Architectural Design

Leveraging the theoretical grounds introduced in Chapter 2 and the related works described in Chapter 3, different model classes were implemented, inheriting the PyTorch `Module` class. The module responsible for initializing the models was implemented using a

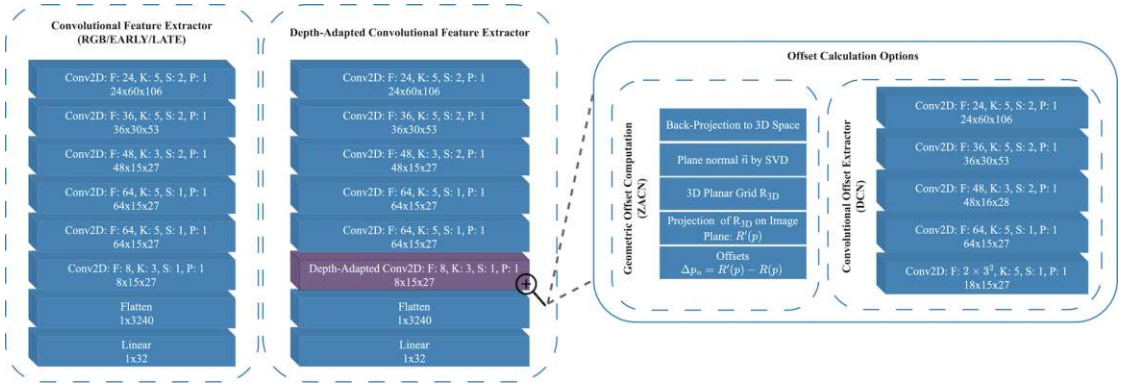


Figure 5.2: A detailed illustration of the layer order in the convolutional heads is provided. The initial feature extractor accepts an RGB input of dimensions 3×120×212. Under the early fusion approach (EARLY), an input of 4×120×212 is processed, whereas the late fusion scheme (LATE) applies the extractor twice, handling RGB and depth inputs of sizes 3×120×212 and 1×120×212, respectively, merging their outputs after the final layer. Additionally, the depth-adapted feature extractor is offered in two variants: (1) DCN, which integrates an offset extraction module, and (2) ZACN [WASD20], which employs geometric offset computations. Notations in the hyperparameter settings are defined as follows: F for filter count, K for kernel dimension, S for stride length, and P for padding. From [CFRG25].

factory pattern [GHJV95]. `BaseModel` encapsulates common functionalities shared by all models, such as standardized initialization, weight configuration, forward propagation, and checkpoint management. Each model type (`EarlyFusionModel`, `LateFusionModel`, `RGBModel`, `DeformableDepthModel`) inherits from the `BaseModel` class, minimizing code duplication while maintaining clear distinctions between architectures. Additionally, the use of enumerations (such as `RNNEnum`, `ExtractorEnum`) standardizes model selection and simplifies configuration handling.

Unimodal Architecture

The implemented **unimodal** architecture is inspired by [LHA+20], having the convolutional head (Figure 5.2) use 6 convolutional layers, the output of which is forwarded to the following RNN. By extracting the most important RGB features, the RNN is used to make predictions based on past memories [CFRG25].

Early Fusion Architecture

The early fusion technique for multimodal extraction uses the depth information as an additional channel in the previously described convolutional head [CFRG25]. The technique placed first amongst counterparts both in [XCG+19] and [MVLÁCBAM24].

Late Fusion Architecture

Similar to [ESS+15], the third architectural technique is **late fusion**. In this two-stream layout, separate CNNs process the RGB and depth data. However, while the first approach finds ways to transform the 1-channel depth input into 3-channel RGB input, this thesis immediately uses the original 1-channel values. The motivation behind this is that the former uses the same pre-trained CNN for each stream (i.e., ImageNet), which was trained on RGB inputs. This is not necessary when the models are trained from scratch.

Depth-Adaptive Fusion Architecture

Finally, the architecture referred to in this thesis and the published counterpart [CFRG25] as *depth-adaptive fusion*. This processes both inputs with Deformable Convolutional Network (DCN)s. This too branches into two choices, as illustrated in Figures 5.1, and 5.2. Both versions employ a single deformable layer. The motivation behind this is that, although it is desirable for the filters to be guided by depth information, it is more computationally efficient to introduce the offsets at a later stage. This thesis aims to prove that later stage offset use is more efficient for the computational constraints on the future deployment platform while still remaining properly influenced by depth cues.

To best implement this decision while maintaining minimal overhead, the `DeformableDepthModel` class is divided at the forward call based on the selected deformable convolution type. In the case of the DCN approach, the offsets are computed by executing `offsets = self.offset_generator(depth)`, where `self.offset_generator` is implemented as a convolutional layer tailored to process depth data. Conversely, when the model type is set to ZACN, the depth-adaptive sampling method is called to compute the offsets. This design decision encapsulates the specific behaviors within a single class while maintaining modularity, as each method is isolated and can be modified independently.

DCN

On the one hand, the depth-adaptive fusion models employ an architecture in which the offsets are learned using a secondary CNN. After experimenting with several PyTorch implementations of DCNs, the implementation available at [Ins24] was chosen. The choice was motivated by both the faster computation time during stress-testing (as opposed to the one contained in the *torchvision* package), and the availability of the Deformably-Scaled Transposed Convolution (DSTC) implementation.

ZACN

On the other hand, models were trained using depth-adaptive sampling [WASD20] in their architectures (described in Subsection 2.2.2). As the implementation of the authors is, at the time of writing, open-source, it was immediately integrated using a convolutional head illustrated in Figure 5.2. Not many changes were necessary to

introduce the implementation into the stack, most being syntax-related or driven by improving computational efficiency. The intrinsic camera parameters used by this method were obtained from the device used during data collection of Section 4.1.

Optional Blocks

Although the hyperparameters of the convolutional layers replicate the best-performing values obtained by [LHA+20], additional configurations were experimented with.

We introduced per-image normalization layers atop the convolutional heads and gave the option to train the models with and without the padding configurations described in Figure 5.2 in order to determine which layout best suits each architecture.

Per-image channel-wise normalization was tested to verify its benefits before feature extraction in the CNN heads. We experimented with min-max normalization given by Equation (5.1) per each channel in the image, Z-score normalization given by Equation (5.2), and min-max followed by Z-score. The normalization strategy can be selected via an enumeration (`NormalizationEnum`).

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5.1}$$

$$z = \frac{x - \mu}{\sigma_{\text{adj}}} \tag{5.2}$$

where:

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N} \tag{5.3}$$

$$\sigma_{\text{adj}} = \max\left(\sigma, \frac{1.0}{\sqrt{N}}\right) \tag{5.4}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}} \tag{5.5}$$

Normalization is introduced as a layer rather than normalizing the dataset image-wise beforehand because the goal is to deploy the models on hardware. This way, streams are automatically normalized frame by frame inside the models.

The optional `PaddingLayer` was introduced due to PyTorch not supporting padding when strides are > 1. The layer calculates the necessary amount of padding on all four sides of the input based on the input shape, kernel size, and stride, then pads the input tensor using zero values.

5.3 Training and Metrics

The training pipeline consists of a primary training loop, with train, validation, and test steps. The methods implement forward propagation, loss computation, backpropagation, and parameter updates. Furthermore, all steps include logging via Tensorboard, while the validation step implements an early-stopping mechanism to avoid overfitting.

The experiments were initialized with different configurations and were run automatically using a shell script. The script systematically iterates over multiple model types (`CfC`, `LTC`, `LSTM`, `LRC`), learning rates, and normalization/padding options. This approach enhances reproducibility and facilitates extensive hyperparameter tuning with minimal manual intervention.

The following arguments are offered at runtime:

- Input mode: unimodal (RGB), multi-modal (RGB-D)
- Fusion technique for multi-modal architectures
- Including/omitting the normalization layer
- Normalization technique to be used in the normalization layer: Min-Max, Z-Score, Min-Max followed by Z-Score
- Including/omitting padding in the convolutional head
- Learning rate
- Seed to be used for reproducibility

The maximum number of epochs was set to 100, and the batch size to 20 [CFRG25]. The reason for selecting this batch size was constrained by the computational resources available. Tensorboard was used during training, validation, and testing [CFRG25].

An early stop [MB89] patience of 3 was introduced for training all models. Therefore, in cases where the validation loss did not improve, the training was stopped early to prevent overfitting of the models. Moreover, the models were saved after each epoch where the validation loss improved.

The optimization algorithm used was Adam [KB14], with $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The value of $\epsilon = 10^{-7}$ was chosen to enhance numerical stability during parameter updates in scenarios with very small gradients. The following learning rates were used in the search for a balance between convergence speed and stability: $10^{-3}, 10^{-4}, 10^{-5}$.

As the task at hand does not involve concrete labeling using classes, quality assessments of classification, such as precision and accuracy, were not used. Instead, with the goal of predicting continuous steering commands based on multi-dimensional input, Mean Squared Error (MSE) given by (2.4) was used as a loss function. Based on the MSE metric, losses closest to 0 are ideal as they represent qualitative predictions [CFRG25].

CHAPTER 6

Results

The following chapter presents the outcome of the training procedures on the various model architectures presented in Section 5.2 and discusses them in detail. As a first approach, it starts with the "passive" evaluation of the model on the recorded test dataset. In continuation, the hardware deployment of the best models is explained and the outcome of the active testing session is presented.

6.1 Open-loop Results

The training procedure described in Section 5.3 is followed. In the *open-loop* setting, the previous prediction of the model does not influence the next input frame (the model is evaluated on a static dataset).

The process began with the training of unimodal RGB models across all possible configurations. The analysis of the validation losses for each model is conducted to find the optimal combination of learning rate, normalization technique, and padding for each RNN architecture.

6.1.1 Validation Analysis Across Learning Rates

For the experiments conducted with a learning rate of 10^{-5} shown in Table 6.1, MinMax normalization with padding emerged as the most promising configuration for most architectures over three different seeds. However, additional experiments were performed at higher learning rates (10^{-4} presented in Table 6.2 and 10^{-3} in Table 6.3) using the same seed to investigate their impact on convergence.

At a learning rate of 10^{-4} (see Table 6.2), certain configurations (e.g., CfC under MinMax normalization with padding) achieved a lower validation loss (e.g., 7.58) compared to other settings. However, these lower losses were typically reached after only 6 epochs, in

Table 6.1: Best validation losses, averaged over 3 seeds for learning rate of 10^{-5}. Loss values are scaled by $\times 10^{-3}$. The symbols 'N' and 'P' denote the normalization technique and whether padding was used for training. 'Epochs' denote the number of epochs that each model trained for until 'early stop' (during the same seed run).

N	P	LSTM	LTC	CfC	LRC	Epochs
None	✗	10.74 ± 1.82	10.61 ± 1.62	10.87 ± 0.92	9.71 ± 1.87	8/11/11/10
	✓	8.78 ± 1.89	10.98 ± 3.41	12.16 ± 6.05	9.48 ± 3.14	11/10/4/5
MinMax	✗	7.28 ± 1.91	11.12 ± 1.26	10.27 ± 1.37	10.93 ± 2.22	20/17/13/9
	✓	8.51 ± 1.30	10.30 ± 0.63	9.44 ± 0.48	8.86 ± 0.65	28/11/9/20
Z-score	✗	12.32 ± 2.73	12.17 ± 3.05	10.47 ± 1.38	8.89 ± 1.55	7/10/29/15
	✓	10.03 ± 2.60	10.69 ± 1.30	10.23 ± 1.56	7.72 ± 1.51	6/14/7/17
Both	✗	10.30 ± 0.72	11.98 ± 2.69	11.27 ± 3.59	8.85 ± 1.08	22/18/8/17
	✓	10.34 ± 2.73	10.31 ± 1.00	10.30 ± 2.17	7.88 ± 1.12	5/13/7/24

Table 6.2: Best validation loss for learning rate of 10^{-4}. Loss values are scaled by $\times 10^{-3}$. 'Epochs' denote the number of epochs that each model trained for until 'early stop' (during the same seed run).

Normalization	Padding	LSTM	LTC	CfC	LRC	Epochs
None	✗	11.03	12.68	11.57	11.90	5/11/8/10
	✓	8.61	32.30	7.58	29.50	10/5/6/4
MinMax	✗	11.27	12.46	12.89	11.27	8/7/6/6
	✓	8.79	32.70	10.78	7.60	9/4/5/15
Z-score	✗	10.74	12.89	9.50	12.62	9/13/10/7
	✓	14.87	32.71	13.66	10.68	5/6/6/7
Both	✗	13.48	14.21	11.13	11.22	5/5/6/10
	✓	11.45	32.63	10.60	10.28	6/4/6/5

contrast to the more gradual training (spanning 9 to 28 epochs) observed with a learning rate of 10^{-5}. As demonstrated in Figure 6.1, the apparent benefit at 10^{-4} is misleading: while the loss initially drops to a very low value at epoch 2, it then becomes highly unstable, with fluctuations and a subsequent spike when early stopping is activated. This behavior underscores that the smooth and consistent convergence achieved with a 10^{-5} learning rate more accurately reflects robust learning dynamics.

When the learning rate was increased further to 10^{-3}, presented in Table 6.3, validation losses across all architectures deteriorated heavily (with losses around 29-32 for LSTM, LTC, and CfC), indicating that such high learning rates hindered effective convergence.

The experiments demonstrate that high learning rates were prone to unstable convergence

Table 6.3: Best validation loss for learning rate 10^{-3}. Loss values are scaled by $\times 10^{-3}$. 'Epochs' denote the number of epochs that each model trained for until 'early stop' (during the same seed run).

Normalization	Padding	LSTM	LTC	CfC	LRC	Epochs
None	✗	29.27	28.14	31.93	27.39	4/4/4/4
	✓	28.83	28.33	31.87	27.29	4/4/4/4
MinMax	✗	29.74	28.35	32.23	27.64	4/4/4/4
	✓	29.68	28.50	32.18	26.55	4/4/4/4
Z-score	✗	29.73	27.11	31.41	23.49	4/4/4/4
	✓	29.50	28.17	32.24	27.91	4/4/4/4
Both	✗	29.68	28.14	31.52	27.50	4/4/4/4
	✓	28.76	27.78	32.24	27.59	4/4/4/4

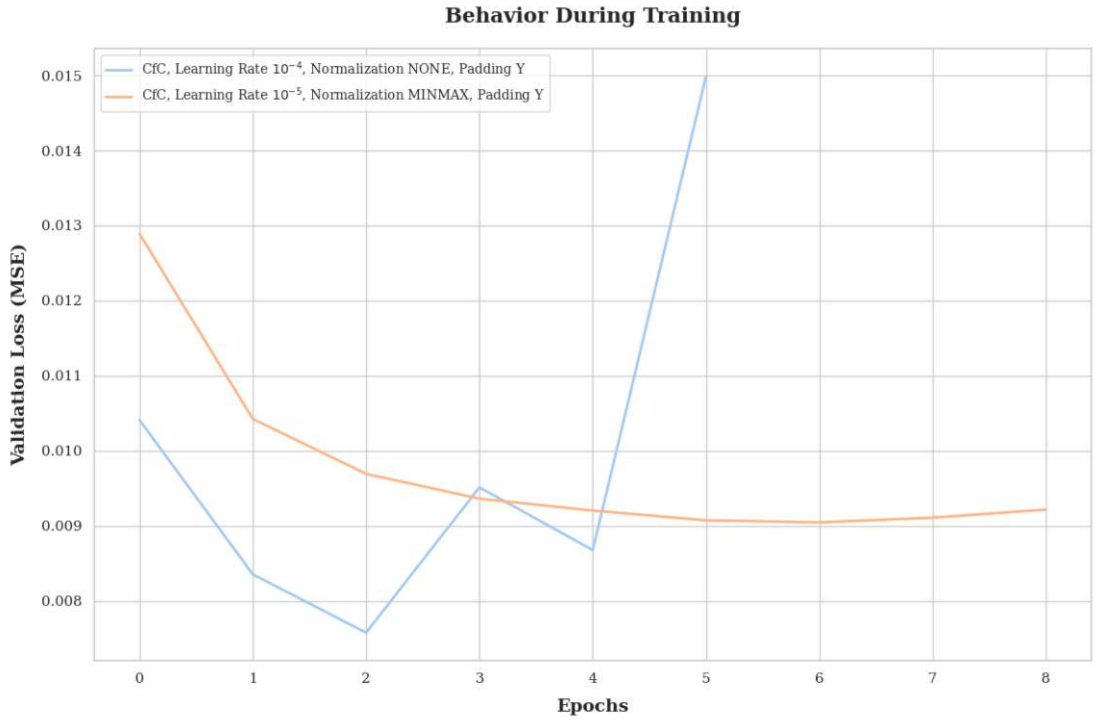


Figure 6.1: Comparison of validation losses for the CfC model using learning rates of10^{-5} and 10^{-4}. The plot shows that the learning rate of 10^{-5} converges smoothly, while the learning rate of 10^{-4} exhibits an erratic trajectory: it reaches a low loss at epoch 2, then fluctuates wildly before spiking as early stopping is triggered.

in a small number of epochs, even though they sometimes yielded lower loss values early on. Thus, the lower learning rate of 10^{-5}, despite requiring more training epochs, appears

more promising due to its consistent and gradual convergence, resulting in lower and less variable validation losses.

6.1.2 Normalization and Padding

The different configurations were evaluated using multiple seeds, revealing key insights into the effects of normalization and padding across the RNN architectures. Among all tested configurations of Table 6.1, MinMax normalization with padding proved to be the most effective technique, consistently yielding lower validation losses across multiple architectures and runs. This trend is particularly evident for CfC, which achieved its best performance with MinMax normalization and padding, recording a validation loss of 9.44 ± 0.48. The relatively small standard deviation in this setting suggests greater training stability, reinforcing the suitability of MinMax normalization for this architecture.

For LSTM, the lowest validation loss was observed in the MinMax normalization without padding setting (7.28 ± 1.91). However, the high standard deviation in this case indicated considerable variability across training runs. To ensure stability, MinMax with padding was selected as the optimal configuration for LSTM (8.51 ± 1.30), as it offered a more reliable trade-off between performance and consistency. This highlights the impact of padding, which, in some cases, can contribute to more predictable training dynamics.

LTC demonstrated minimal sensitivity to normalization techniques compared to the other architectures, but it still followed the general trend where MinMax with padding provided the lowest validation loss (10.30 ± 0.63). While this performance did not dramatically differ from other settings, its smaller deviation made it a preferable choice. Notably, Z-score normalization resulted in consistently higher losses for LTC, indicating that this approach might not be well-suited for this architecture.

For CfC, the results demonstrated a clear preference for MinMax normalization with padding, achieving the lowest validation loss at 9.44 ± 0.48. This configuration not only provided the best overall performance but also exhibited a remarkably low standard deviation, indicating consistent training behavior across different seeds. In contrast, other normalization techniques, such as Z-score and Both, resulted in higher losses and greater variability. Notably, when no normalization was applied, CfC's performance suffered, with a validation loss of 10.87 ± 0.92 without padding and an even higher deviation when padding was introduced. This highlights the importance of selecting an appropriate normalization method, as improper preprocessing can lead to instability and degraded performance, particularly for architectures like CfC that may be more sensitive to input scaling variations.

The performance of LRC shows that the MinMax normalization with padding is still among the top-performing configurations. Although Z-Score with padding and Both with padding have low MSE values, their standard deviation is very high, exceeding 1. However, MinMax technique shows a deviation of only 0.65, which makes it the better choice to train LRC architectures.

Table 6.4: Averaged MSE test losses computed over three independent seeds. The models were trained and evaluated on validation and test subsets using the recorded dataset, with loss metrics scaled by $\times 10^{-3}$. The bold text emphasizes the two best-performing recurrent models for each feature extractor. From [CFRG25].

	LSTM	LTC	CfC	LRC
EARLY	$\mathbf{7.53 \pm 0.49}$	11.73 ± 3.82	12.60 ± 4.42	$\mathbf{8.23 \pm 1.71}$
LATE	$\mathbf{5.93 \pm 0.77}$	9.93 ± 1.02	8.97 ± 1.63	$\mathbf{7.80 \pm 0.92}$
ZACN	$\mathbf{8.27 \pm 1.22}$	10.27 ± 0.90	9.53 ± 1.17	$\mathbf{8.07 \pm 1.10}$
DCN	$\mathbf{8.77 \pm 1.63}$	22.50 ± 9.75	11.65 ± 1.85	$\mathbf{9.37 \pm 1.88}$
RGB	$\mathbf{9.57 \pm 2.37}$	13.30 ± 1.91	$\mathbf{11.87 \pm 1.74}$	12.00 ± 2.14

Beyond individual architectures, the overall comparison of normalization strategies indicates the robustness of MinMax normalization when paired with padding. While other techniques like Z-score and no normalization occasionally yielded competitive results, they were generally associated with higher variability and inconsistent performance.

In summary, the validation experiments using RGB input revealed that:

- Although certain configurations at a learning rate of 10^{-4} exhibited lower validation losses, these were often associated with erratic convergence during training, suggesting that the models may not generalize as well.
- A learning rate of 10^{-3} consistently produced higher validation losses, highlighting the risk of instability and insufficient convergence at excessively high learning rates.
- The learning rate of 10^{-5} resulted in more stable and consistent training across different architectures.

Based on these observations, MinMax normalization with padding at a learning rate of 10^{-5} is selected as the optimal configuration for training the multimodal models.

6.1.3 Evaluation on the Test Dataset

The open-loop test results, as reported in Table 6.4, reveal an interesting behavior for the RGB modality. While the two best-performing recurrent models for most feature extractors are LSTM and LRC, RGB exhibits an exception where CfC delivers competitive performance, despite being outperformed by LRC in other modalities. Specifically, LSTM achieves a test loss of 9.57 ± 2.37 compared to CfC's 11.87 ± 1.74. This divergence suggests that the architectural differences impact the way spatial and temporal information are integrated. The lower loss obtained by LSTM on the RGB dataset indicates that its dynamics may be better suited to capturing the underlying features of the RGB input, whereas CfC, which relies on a different recurrence mechanism, might require additional tuning or a different preprocessing strategy to fully leverage the RGB modality.

Table 6.5: Maximum pairwise difference in steering angle [rad] predictions for the models over the first lap in one of the test dataset recordings (with the same seed). The maximum difference between two successive steering commands given by the human expert during the lap was **0.081**.

	LSTM	LTC	CfC	LRC
EARLY	0.125	**0.043**	0.123	**0.078**
LATE	0.052	**0.039**	0.143	**0.044**
ZACN	0.129	**0.077**	0.126	**0.061**
DCN	**0.067**	0.072	0.103	**0.048**
RGB	0.072	**0.026**	0.057	**0.031**

Table 6.5 can be used to interpret the smoothness of the steering commands. Ideally, a model that accurately replicates human behavior should produce steering commands with relatively small differences between successive values, mirroring the human expert's maximum difference of 0.081 rad. The revised results show that models exhibit a range of maximum differences, with some configurations achieving very low values (RGB_{LTC} at 0.026 rad and RGB_{LRC} at 0.031 radians). On the one hand, this could suggest highly stable predictions. Other configurations, such as LATE_{CfC} at 0.143 rad and DCN_{CfC} at 0.103 rad, provide sparser predictions, which could indicate a more erratic control behavior.

On the other hand, too small a difference may suggest the inability to steer more when necessary. Essentially, if a model's maximum difference is significantly smaller than the human reference, this could imply an inability to react to sudden but necessary steering changes, such as sharp turns or obstacles. Therefore, these values must be complemented with the MSE values from Table 6.4. A reliable model should be able to produce smooth transitions without being overly slow in response.

6.2 Closed-loop Results

After training and evaluating the models on the passive dataset, it became interesting to explore how they would adapt to a closed-loop setting in which the controller is deployed in the real world.

6.2.1 Selected Models

Given that the total number of models amounted to twenty (five distinct feature extractors, each combined with one of four possible RNN options), it was considered practical to select only the models that had the best performance on the dataset for further evaluation.

Models were ultimately selected for closed-loop testing on the basis of the lowest test losses and minimal standard deviations observed over three seeds from Table 6.4, as these metrics are considered to provide a more robust and statistically sound measure of overall

Table 6.6: Summary of the closed-loop testing, with results organized according to different feature extractors. For each feature extractor, the two most effective RNN options were deployed. 'Success' refers to the successful completion of five consecutive laps on the test track. From [CFRG25].

Feature Extractor	RNN	Success
EARLY	LSTM	✓
	LRC	✗
LATE	LSTM	✗
	LRC	✗
ZACN	LRC	✓
	LSTM	✗
DCN	LSTM	✓
	LRC	✗
RGB	LSTM	✗
	LRC	✗

performance. It is believed that the consistency across multiple training runs offers a far more reliable indication of a model's suitability for real-world deployment than the maximum steering difference observed over a single test lap from Table 6.5. Nevertheless, the maximum difference metric is maintained as an important check, and its predictive value regarding driving smoothness will be revisited during real-world testing.

In order to ensure a balanced and comprehensive assessment, test groups were structured to account for both the feature extractors and the RNNs. Specifically, the two best-performing recurrent models were chosen for each feature extractor, while conversely, the two most effective feature extractors were identified for each recurrent model across all tested seeds. The results of the closed-loop experiments are systematically categorized in Tables 6.6 and 6.7.

Many of the trained models encountered a considerable degree of difficulty when subjected to the real-world setup, with only a subset demonstrating the capability to complete five continuous laps without experiencing a crash. The specific locations at which crashes occurred have been marked in Figure 6.2. In comparison to the maps built for dataset recordings, which are described in Table 4.1, the map is 24.25m long (shorter on average by 9.07m). The map has average left and right distances of 0.5m and 0.55m, respectively, which is very close to the dataset maps, which have an average width of 0.562m and 0.53m. The human driver recorded laptimes of $\approx$ 27.93s in ACW direction (8.569s faster than the average 36.492s lap time on the dataset). These metrics enforce the balanced design of the circuit for closed-loop testing. Notably, the human driver's faster lap time was likely due to the significantly shorter track length, all while maintaining a level of navigational difficulty comparable to that of the dataset maps.

Table 6.7: Summary of the closed-loop testing, with results organized according to different recurrent models. For each RNN, the two best-performing feature extractor options were selected based on the test results obtained from the recorded passive dataset. 'Success' refers to the successful completion of five consecutive laps on the test track. From [CFRG25].

RNN	Feature Extractor	Success
LSTM	LATE	✗
	EARLY	✓
LTC	LATE	✗
	ZACN	✗
CfC	LATE	✓
	ZACN	✓
LRC	LATE	✗
	ZACN	✓

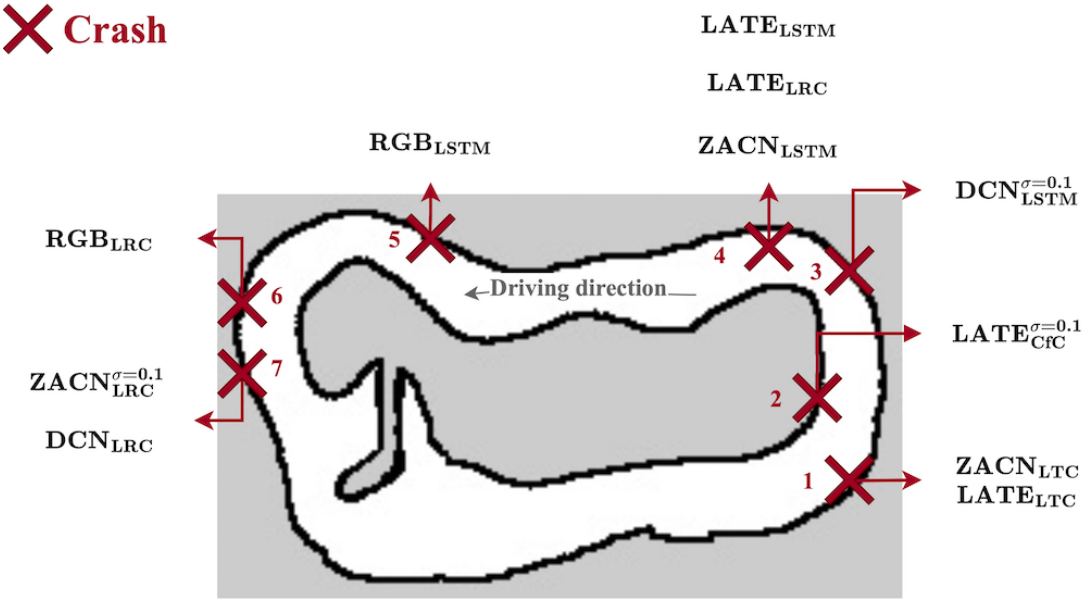


Figure 6.2: The map of the test circuit used in active closed-loop evaluations. Critical turning points that led to irrecoverable crashes are marked with an 'X', along with annotations that identify the responsible models. From [CFRG25].

Of particular note is the observation that the LATE feature extractor achieved success only when paired with the CfC model, despite the fact that this particular recurrent model had not been among the two highest-performing models for the LATE type in the passive open-loop evaluation (Table 6.4). This outcome shows the sim-to-real gap: how performance in the open-loop training setup can diverge from deployment on hardware.

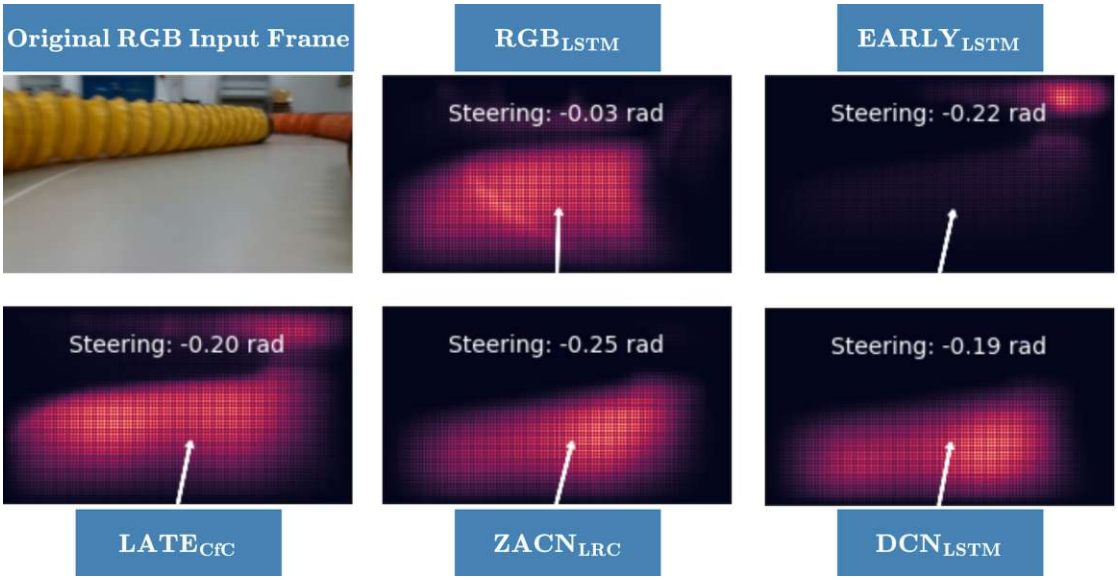


Figure 6.3: Attention maps produced during closed-loop testing, evaluated for the RGB$_{\text{LSTM}}$ model and the models that achieved full autonomy across five consecutive laps. The chosen frame corresponds to the moment immediately preceding the crash of the RGB$_{\text{LSTM}}$ model at landmark 5, as indicated in Figure 6.2.

6.2.2 Visualizing Models' Attention

To better understand the underlying mechanisms governing model behavior, the focus was shifted to identifying the specific regions of the input to which model attention was most strongly directed. Offline analysis of the recordings for each run was conducted using VisualBackProp [BCC+16]. Bright regions in the attention maps in Figure 6.3 indicate areas of focus, and overall suggest that each model has a unique attention pattern.

Interestingly, although the EARLY$_{\text{LSTM}}$ model tended to maintain its attention on more distant regions, the remaining models focused primarily on the road immediately ahead.

Furthermore, the LATE$_{\text{CfC}}$, ZACN$_{\text{LRC}}$, and DCN$_{\text{LSTM}}$ models demonstrated a tendency to direct attention toward the right side of the road.

In contrast, the RGB$_{\text{LSTM}}$ model distributed its focus evenly in front without concentrating sufficiently on upcoming turns. It is possible that this failure to adequately anticipate turns contributed significantly to the eventual crash of the RGB$_{\text{LSTM}}$ model.

6.3 Robustness to Noise

The models that successfully completed five consecutive laps without crashing were selected to evaluate robustness under noisy conditions. These included EARLY$_{\text{LSTM}}$, LATE$_{\text{CfC}}$, ZACN$_{\text{LRC}}$, and DCN$_{\text{LSTM}}$. To assess their resilience, additional Gaussian

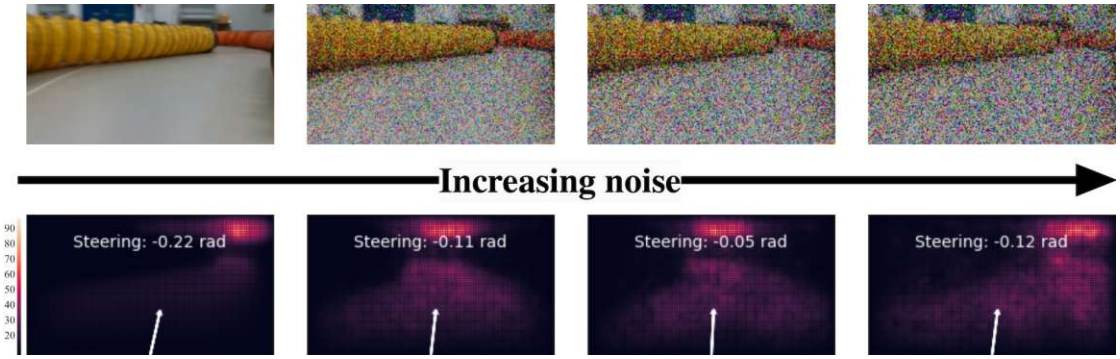


Figure 6.4: Attention maps from closed-loop recordings are shown for noise-free input as well as for Gaussian noise levels with variances of $\sigma = 0.1$, 0.2, and 0.3. Evaluations on the top-performing $\text{EARLY}_{\text{LSTM}}$ network reveal that its focus remains consistently directed at the same region despite increasing noise. From [CFRG25].

noise, characterized by a mean of 0 and a variance of 0.1, was introduced into the input stream.

The experiment had an intriguing result, with $\text{EARLY}_{\text{LSTM}}$ being the only model able to maintain full autonomy when subjected to sensor noise (over both RGB and depth streams). The other models failed, resulting in crashes occurring at Points 2, 7, and 3, as indicated in Figure 6.2.

The series of images in Figure 6.4 illustrates the effects on $\text{EARLY}_{\text{LSTM}}$'s attention of both a noiseless input and an input that has been increasingly perturbed by Gaussian noise with variances of $\sigma = 0.1, \sigma = 0.2$, and $\sigma = 0.3$, respectively. For the visualizations, the original perturbed stream and predictions were used.

The $\text{EARLY}_{\text{LSTM}}$ model withstood all increasing levels of input noise without a degrading in performance. Notably, the model maintains its focus on a stable region of interest consistently, despite the introduction of perturbations. This constancy in the region of interest has been interpreted as a strong indicator of the model's resilience to environmental uncertainties.

6.3.1 Quantifying Attention

Although the models' attention maps are visually compelling, a numerical estimate helps establish the extent to which they were altered under increased noise levels.

The Structural Similarity Index (SSIM) [WBSS04] was used to quantify these alterations. SSIM assigns each pairwise comparison of images a score in the $[0, 1]$ interval. In this case, it represents an indication of the extent to which the network's focus remained unaffected by noise. Higher SSIM values, approaching 1, relate to the model's sustained attention.

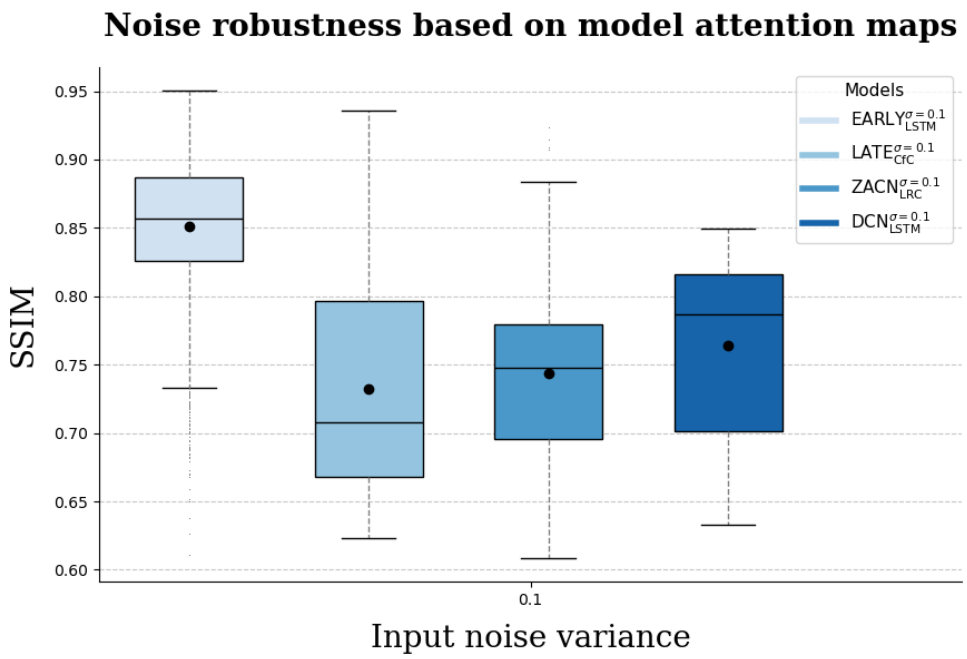


Figure 6.5: Calculated SSIM metrics for the $\text{EARLY}_{\text{LSTM}}$, LATE_{CfC}, ZACN_{LRC}, and DCN_{LSTM} networks, assessing the stability of their attention mechanisms amid noise. The $\text{EARLY}_{\text{LSTM}}$ network achieved the best scores, reflecting its robustness to noisy inputs compared to the other models. From [CFRG25].

When the $\text{EARLY}_{\text{LSTM}}$ model was subjected to increased noise conditions, its resulting SSIM values were observed to be significantly higher than those of other models, as has been illustrated in Figure 6.6.

With the purpose of further testing $\text{EARLY}_{\text{LSTM}}$'s robustness, two more experiments were conducted where the noise variance was incrementally adjusted in 0.1 steps. Remarkably, the model exhibited high robustness to these heightened perturbations.

Particular attention was given to how the model maintained focus on key input features without being distracted by noise. Figure 6.4 shows the attention maps obtained through VisualBackProp [BCC+16].

After examining the corresponding SSIM values presented in Figure 6.6, it was determined that, even under conditions of considerable noise, the $\text{EARLY}_{\text{LSTM}}$ model succeeded in preserving an impressive degree of attention stability. Specifically, the similarity scores were measured at 86%, 75%, and 67%, respectively, for the cases of $\sigma = 0.1, \sigma = 0.2$, and $\sigma = 0.3$.

Though there is a decline in these values as noise levels increased, the rate at which they decreased was not strictly linear; rather, a deceleration in the rate of decline with reductions of -14%, -11%, and -8% was observed in succession.

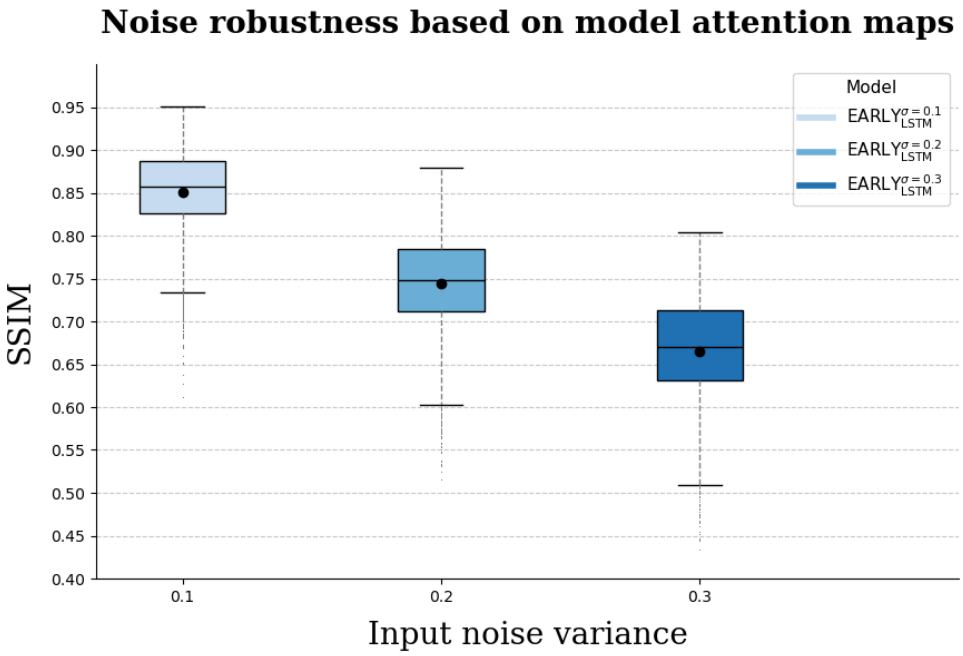


Figure 6.6: Assessment of noise robustness during closed-loop trials was conducted using the EARLY$_{\text{LSTM}}$ network, which maintained complete autonomy even with increased noise levels. This figure presents SSIM indices comparing attention maps from noise-free conditions to those with Gaussian noise variances of $\sigma = 0.1$, 0.2, and 0.3. From [CFRG25].

This finding suggests that while the influence of noise was not entirely negligible, the model's ability to sustain stable attention was preserved to a considerable extent, even in the presence of substantial perturbations.

This suggests that this robustness may play a most significant role in enabling the model to maintain consistent performance under the challenging and unpredictable conditions of noisy environments.

6.4 Comparison with Human Expert Driver

The success of the EARLY$_{\text{LSTM}}$ model under increasingly demanding conditions was followed by conducting a comparison between the model's behavior and that of the expert human driver who had controlled the *roboracer* during the data collection sessions. To this end, a driving session was held in which the driver completed five continuous laps on the same test track.

6.4.1 Commanded Steering Angles

Firstly, it was necessary to account for the discrepancy in steering commands frequencies, as the human driver's data were recorded at $30fps$ while the model's predictions were issued at a lower frequency. This issue, which is discussed in greater detail in Section 6.5, was addressed by estimating the effective frame rates. Specifically, the number of camera topic messages in the rosbags was compared to the number of predictions published on the */drive* topic during recordings. Using the estimated fps, the model predictions were interpolated to a common timeline shared by both the human and the model data.

Although the steering time series were aligned on a common timeline, matching the values was essential to calculate comparative metrics. Given that both the human driver and the model navigated the same track for the same amount of time and under the same speed cap, their steering commands are inherently expected to exhibit similar patterns. Dynamic Time Warping (DTW) [Sak78, Vin68] is an alignment-based metric that is used to assess the similarity between time series as the sum of the distances between matched features. As opposed to just Euclidean distance matching, the DTW method accounts for temporal distortions and is able to assign several matching features from the base series to one in the other series, obtaining the best alignment between similar time series. In [Tav21], DTW is formulated as the following optimization problem:

$$DTW_q(x, x') = \min_{\pi \in \mathcal{A}(x,x')} \left(\sum_{(i,j)\in\pi} d(x_i, x'_j)^q \right)^{\frac{1}{q}} \tag{6.1}$$

where x, x' are the time series, q is the L_q norm parameter, π is the alignment path, $\mathcal{A}$ is the set of admissible paths.

The time series used by DTW must have a matched start and end index, which is done in this thesis in the previous step during interpolation. The solution to the optimization problem is solved using dynamic programming and further detailed in [Tav21], resulting in an $O(mn)$ algorithm, where m, n are the lengths of the series. The implementation used in this thesis is from the *DTAIDistance* package, available at [DTA25].

Successful models

As can be seen in Figure 6.7, $\mathbf{EARLY_{LSTM}}$ shows slight overshooting tendencies when taking turns. The magnitude of the overshooting is consistent (although not identical) over the 5 laps, with no particular bias to turn directions. Compared to the commands issued by the expert driver, the plot reflects the computed RMSE value of 0.0323, displayed in Table 6.8. $\text{EARLY}_{\text{LSTM}}$ had the lowest latency average and standard deviation, 5ms away from the second lowest, which explains its incredibly fast reaction times and success even under very noisy input. The approach to directly include depth in the convolutional head significantly cuts down computational costs, which is high in all other RGB-D models. Moreover, the computed maximum steering differences also reflect the overshooting of

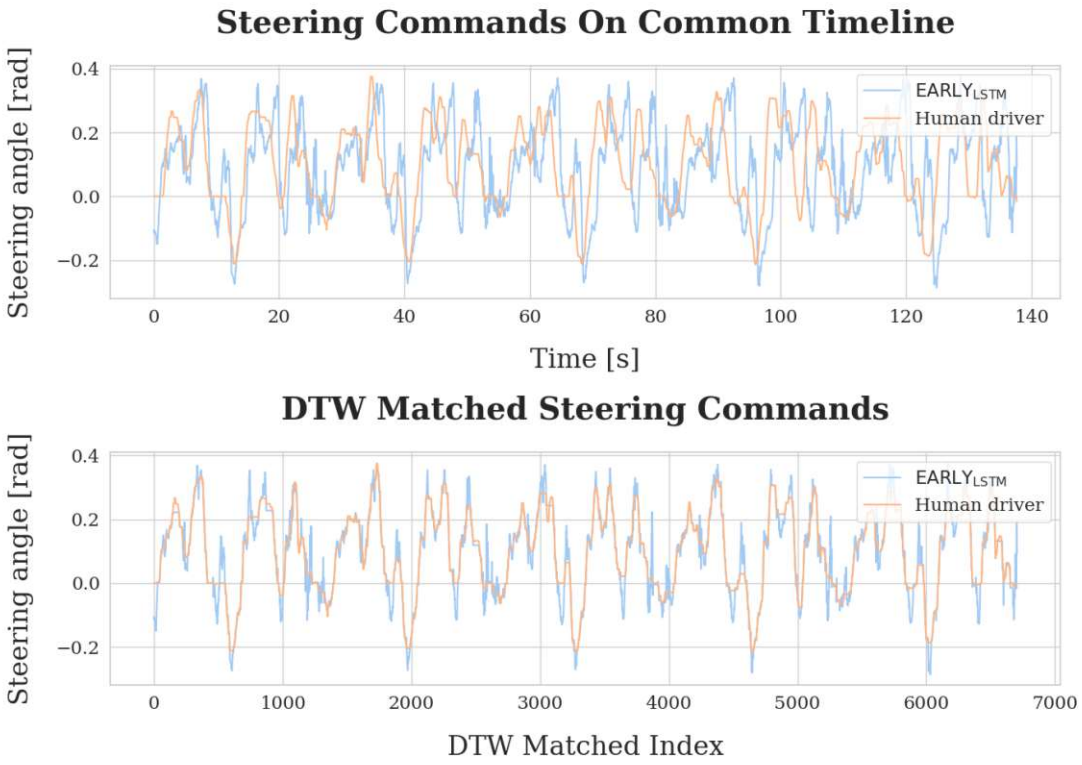


Figure 6.7: Comparison between the steering commands predicted by the EARLY$_{\text{LSTM}}$ model and those of the human expert driver.

the model and the more consistent driving style of the human driver. The noticeable spikes in the interval around DTW Matched Indices 500-800 overlap with the interesting behavior illustrated by the trajectories in Figure 6.15. In this window, while the human driver goes straight on the middle of the track (steering angle 0 radians), the model sways slightly right, then left (following the outer pipeline).

The **LATE$_{\text{CfC}}$** commanded steering values are illustrated in Figure 6.8. Interestingly, the model exhibits a high tendency to overshoot in left turns. However, on the same interval window, where EARLY$_{\text{LSTM}}$ wobbles (500-800 matched indices), LATE$_{\text{CfC}}$ keeps more closely to steering 0 radians. When looking at the plotted trajectory of the model in Figure 6.16, it is evident that it is the only model that keeps more to the middle of the track, similar to the human driver. Table 6.9 reflects that this model has the lowest D_{max}, which coincides with the plotted steering commands. Moreover, it also has the second lowest RMSE value of all successful models. Interestingly, the latency is not very far from the other LATE models that failed the full autonomy tests.

ZACN$_{\text{LRC}}$ exhibits smooth steering behavior and a tendency to keep right, as shown in Figure 6.9. This is evident in its trajectory in Figure 6.16, driving very close to the

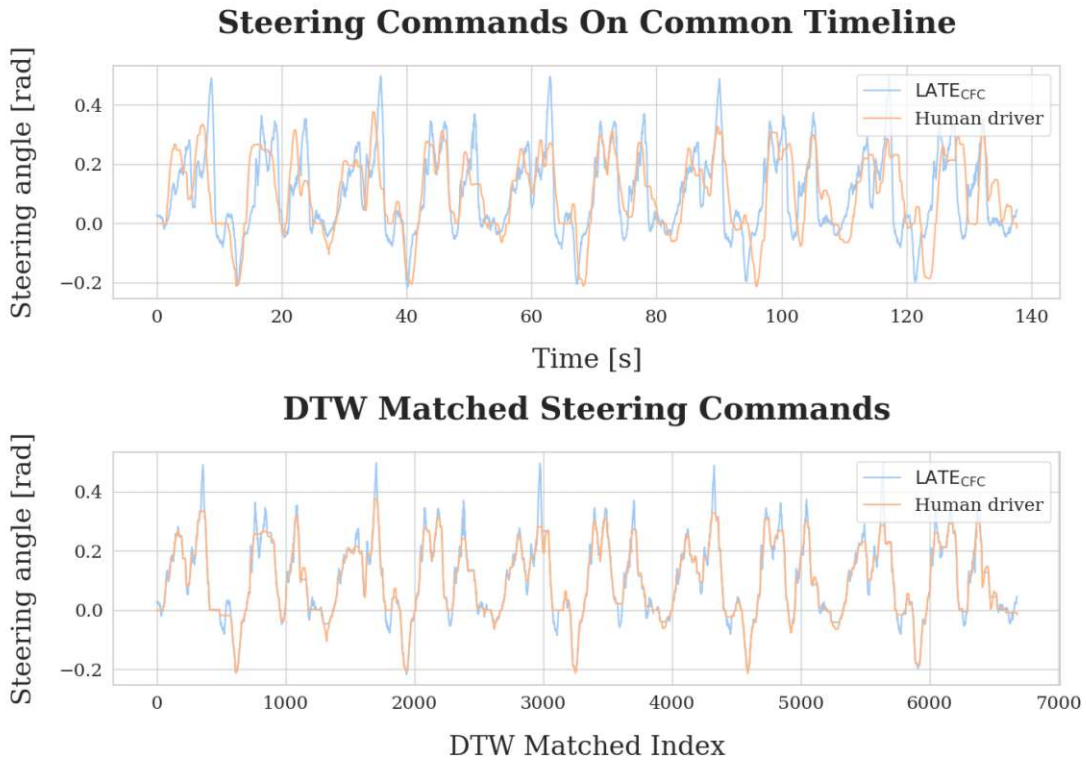


Figure 6.8: Comparison between the steering commands predicted by the LATE_{CfC} model and those of the human expert driver.

outer pipeline after the first turn. Remarkably, the model had the highest latency at 20.54ms in Table 6.8, but the lowest D_{max} and RMSE of all successful models. The latency is attributed to the deformable convolution operation in its last layer. Although not immediately apparent from the trajectory in Figure 6.16, the commanded steering values do overlap very often with the human driver's, which could explain the low error value. The shift in trajectory is caused by the model's rightward bias in driving style. Although the overshooting magnitude is not high enough to affect the RMSE, it alters the trajectory.

The upper plot in Figure 6.10 shows the human driver's commands aligned to the ones of the $\mathbf{DCN_{LSTM}}$ model. The commanded steering values have smooth transitions, which is also evident in the recorded D_{max} value in Table 6.8 (the second lowest value). However, the calculated RMSE also reflects the overshooting behavior of the model at every turn. Its trajectory is very similar to ZACN_{LRC}, prone to follow the outer pipeline very closely, with a rightward driving bias. With a latency twice that of $\text{EARLY}_{\text{LSTM}}$, the second convolutional head necessary for offsets computation seems to be computationally faster than its ZACN counterpart (2ms difference on average).

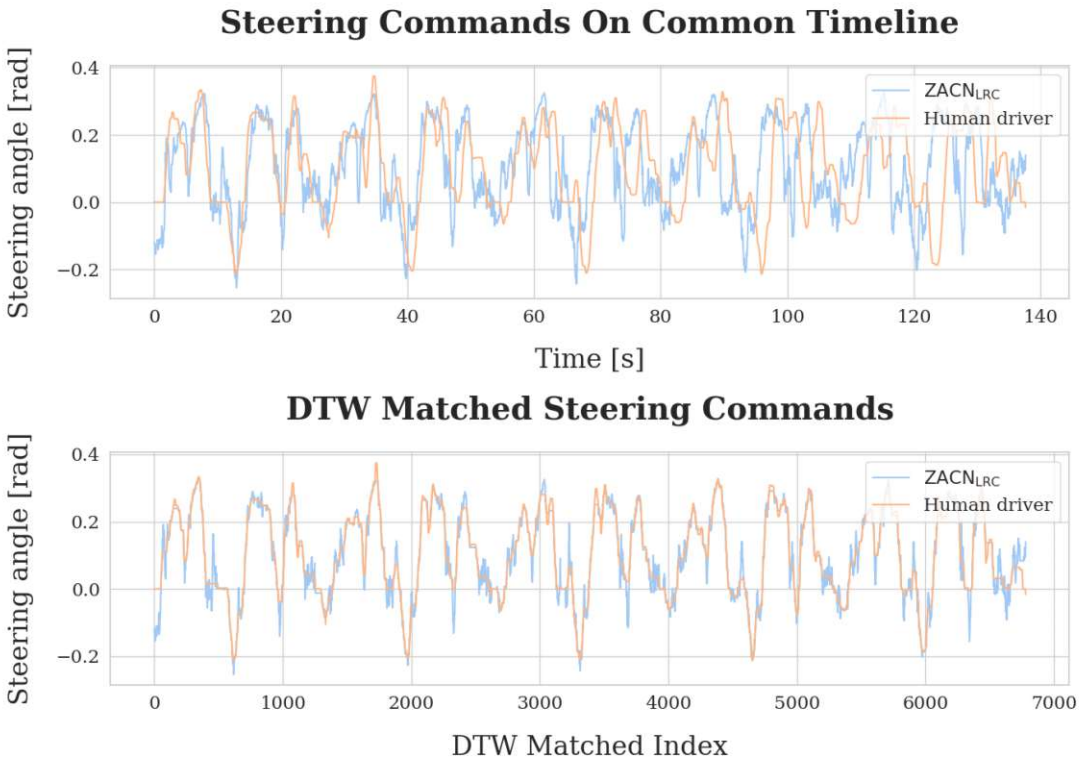


Figure 6.9: Comparison between the steering commands predicted by the ZACN_{LRC} model and those of the human expert driver.

Behaviors prior to crashing

Figure 6.11 shows the commanded steering angles during the recording of $\mathbf{LATE_{LTC}}$ and $\mathbf{ZACN_{LTC}}$ models. Both models crashed in the same spot, very early on the circuit (**position 1**). The behaviors of the two models differ, the LATE fusion LTC is smoother in predictions than its ZACN counterpart, as shown also in Table 6.9 in the D_{max} values of each. However, ZACN seems to follow the human commands more closely, although not with the same magnitude. Judging by the place of the crash (at the beginning of the first turn) and by the commanded steering angles, neither of the two models was able to anticipate the turn in time. They both start steering very late, with an RMSE to the human driver of 0.0851 for the LATE_{LTC} and 0.0884 for ZACN_{LTC}. The error values are similar to the ones the models achieved during open-loop testing in Table 6.4 (when the squared root is applied). Although the latencies registered for each of the two models are similar to some registered by successful models, given the discrepancy in steering commands (compared to the human driver's), the failure can be attributed to computational overload. One cause can be that the late model uses two convolutional heads to extract features from RGB and depth streams in parallel. As such, the only successfully tested LATE model was using a CfC backbone, while the ZACN models

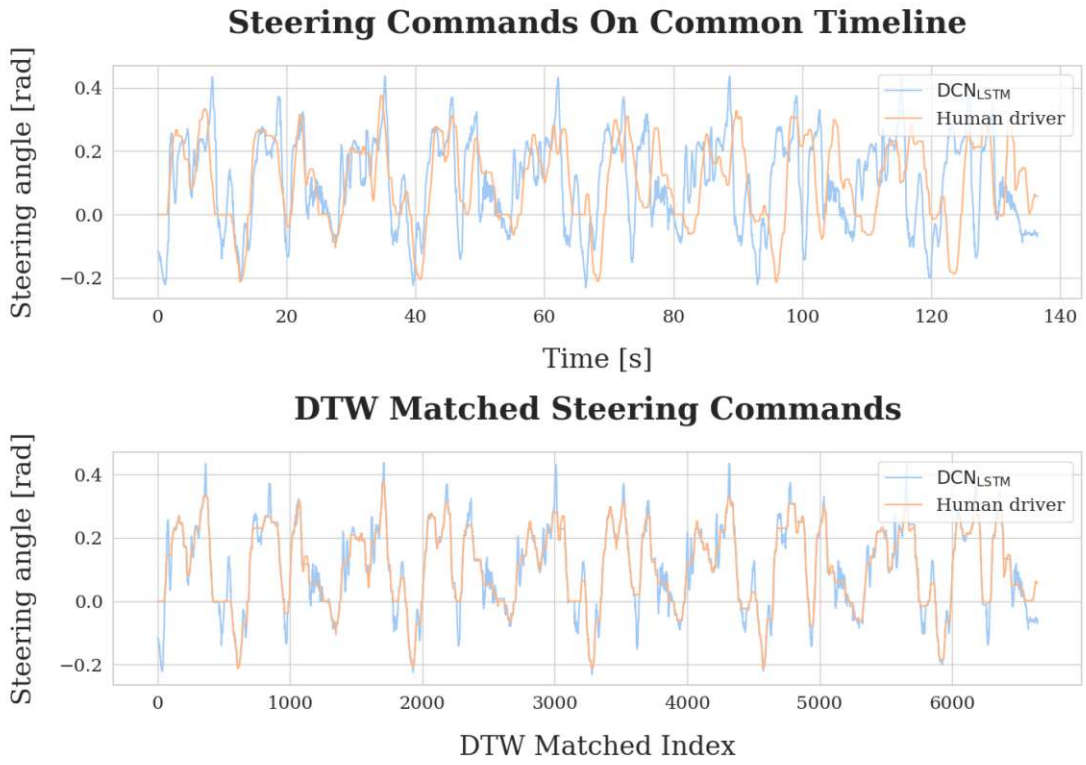


Figure 6.10: Comparison between the steering commands predicted by the DCN_{LSTM} model and those of the human expert driver.

using CfC and LRC both achieved full autonomy, which can indicate that they are more computationally lightweight than LTC on such a resource-constrained platform.

The commanded steering values for the models that crashed at **position 4** on the testing circuit are shown in Figure 6.12. The two **LATE** fusion models achieved the best losses on the test dataset (Table 6.4) and show similar latencies in Table 6.8. Although the **LSTM** model has a D_{max} almost identical to the human's on the portion up to the crash position, its **LRC** counterpart has a very low maximum difference. Moreover, all LRC backbones display the same trend of very close steering predictions. However, when aligned on a common timeline and also matched with the human driver commands, the LATE models that crashed at position 4 were steering right instead of left. Given that the area on the circuit represents the exit coming from a right turn, it seems probable that this behavior was caused by computational delay, given that these models use more complex feature extractors. The **ZACN** model, although close to human behavior, overshoots when steering outside the turn, which causes its trajectory to be off-course. Overall, their predictions seem more influenced by inputs received at past locations (during the right turn) than the present.

The RGB models that were deployed crashed before and after the second turn in the

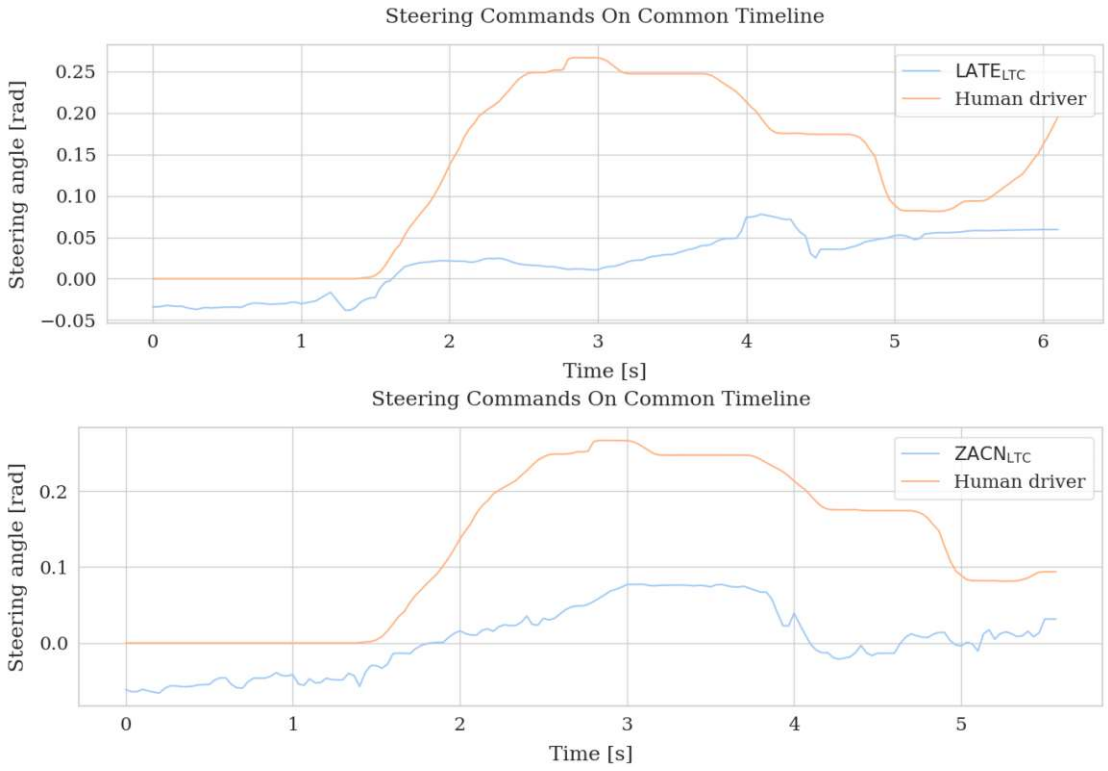


Figure 6.11: Common timeline comparison between the steering commands predicted by the LATE_{LTC} and ZACN_{LTC} models and those of the human expert driver, up to the irrecoverable crash at position 1 in Figure 6.2.

map, at **positions 5 and 6** shown in Figure 6.2. Assessing the predictions given by the models in Figure 6.13, the LRC model has considerably fewer spikes, which can be attributed to its being able to take the turns better than its LSTM counterpart. Tables 6.8 and 6.9 reveal that the RGB_{LRC} model had the highest standard deviation in latency compared to all other models, which is not the case for any other LRC-based models using multimodal feature extractors. Moreover, the higher fluctuations of the RGB_{LSTM} are also reflected in the D_{max} values. Both unimodal models make highly incompatible predictions given the map location: the LSTM model fails to steer left in time to enter the turn, while the LRC model perturbs its own trajectory by suddenly steering left, thus crashing into the wall. Both models seem to be unable to react to sudden changes in the circuit. The failure of the unimodal models was also foreseen by the test results in Table 6.4, where all RGB models scored higher MSE values than the best multimodal models.

The $\mathbf{DCN_{LRC}}$ model crashed very late, at **position 7** on the circuit. While exiting the second turn, it is obvious in Figure 6.14 that the steering commanded by the model suddenly diverges from the human counterpart. Both latency and RMSE values are on the higher side in Table 6.8. The high error is reflected in the steering plots in sudden

Table 6.8: Quantifiers of model behavior during closed-loop testing, grouped by feature extractors. 'S' denotes successful completion of all laps, 'Latency' is the mean and standard deviation of recorded latency during testing, 'D_{max}' is the maximum difference recorded between two successive steering commands (radians), and RMSE is calculated for all steering commands interpolated over a common timeline for the model and human driver. D_{max}^{Human} is calculated for the same driving interval as the model counterpart (i.e., up to the crashing point).

Feature Extractor	RNN	S	Latency [ms]	D_{max}^{Model} [rad]	D_{max}^{Human} [rad]	RMSE
EARLY	LSTM	✓	8.26 ± 7.86	0.1918	0.0544	0.0323
	LRC	✗	8.61 ± 7.51	0.2465	0.2525	0.0916
LATE	LSTM	✗	12.66 ± 11.63	0.2996	0.2999	0.1519
	LRC	✗	13.38 ± 7.98	0.0616	0.3014	0.0455
ZACN	LRC	✓	20.54 ± 11.97	0.1394	0.0544	0.0246
	LSTM	✗	19.77 ± 12.55	0.1209	0.3015	0.0396
DCN	LSTM	✓	18.55 ± 12.11	0.1497	0.0544	0.0339
	LRC	✗	16.82 ± 9.32	0.092	0.159	0.0630
RGB	LSTM	✗	8.78 ± 7.66	0.1151	0.0323	0.0622
	LRC	✗	11.27 ± 15.939	0.0627	0.0323	0.026

spikes right before or during an existing maneuver: around DTW matched index 380, while increasingly steering left, the model makes a sudden rightwards prediction, yet recovers immediately, at indices 580 and 630, the model also abruptly steers left during a right steering maneuver. This behavior ultimately leads to an irrecoverable crash when the model fails to finish the turn exit, continuing to steer in the opposite direction. Although the steering values are smooth (D_{max} is 0.092), as anticipated in 6.2.1 when interpreting Table 6.5, too small of a D_{max} leads to inability steer more when actually needed, especially compared to human behavior with D_{max} 0.159.

6.4.2 Trajectories

The movements of the model and the human driver were tracked in the video recordings from a ceiling-mounted dome camera via object tracking. The implementation is as follows: the *roboracer* is selected in the first frame of the videos, then, using template matching from OpenCV [Bra00], the object is tracked frame by frame, and a trajectory is drawn on the image.

The results of this comparative analysis with the best performing model are shown in Figure 6.15. Interestingly, the side-by-side comparison shows that while the trajectory produced by the $\text{EARLY}_{\text{LSTM}}$ model is smooth, it does not entirely replicate the driving

Table 6.9: Quantifiers of model behavior during closed-loop testing grouped by RNN backbone. The other table headers are as in Table 6.8.

RNN	Feature Extractor	S	Latency [ms]	D_{max}^{Model} [rad]	D_{max}^{Human} [rad]	RMSE
LSTM	LATE	✗	12.66 ± 11.63	0.2996	0.2999	0.1519
	EARLY	✓	8.26 ± 7.86	0.1918	0.0544	0.0323
LTC	LATE	✗	13.67 ± 7.80	0.0210	0.0203	0.0851
	ZACN	✗	21.20 ± 14.87	0.0228	0.2030	0.0884
CfC	LATE	✓	13.85 ± 9.25	0.1029	0.0544	0.0251
	ZACN	✓	21.00 ± 12.22	0.2925	0.0544	0.0352
LRC	LATE	✗	13.38 ± 7.98	0.06165	0.3014	0.0455
	ZACN	✓	20.54 ± 11.97	0.1394	0.0544	0.0246

Table 6.10: Quantifiers of model behavior during closed-loop testing, with noised inputs, of variance level 'σ'. The other table headers are as in Table 6.8.

Model	σ	S	Latency [ms]	D_{max}^{Model} [rad]	D_{max}^{Human} [rad]	RMSE
$\text{EARLY}_{\text{LSTM}}$	0.1	✓	8.705 ± 21	0.2791	0.0544	0.0417
	0.2	✓	7.4514 ± 19.912	0.2204	0.0544	0.0371
	0.3	✓	4.987 ± 11.47	0.2132	0.0544	0.0391
LATE_{CfC}	0.1	✗	14.5 ± 33.47	0.0860	0.4306	0.05932
ZACN_{LRC}	0.1	✗	14.36 ± 12.1	0.1961	0.149	0.0652
DCN_{LSTM}	0.1	✗	15.18 ± 18.73	0.09	0.3545	0.0517

style of the human driver. This discrepancy may be attributed to the expert's extensive driving experience.

Although the model's training included five distinct tracks, with a variety of data (as argued in Section 4.2.1), introducing more track layouts into the training dataset could enhance generalization capabilities. This would enable the model to more closely approximate the driving behavior demonstrated by a human expert.

6.5 Impact of Frame Drops on Model Performance

The analysis of the recorded rosbags and model predictions indicates that the model experienced frame drops during the sessions. The frame rate loss is approximately seven frames per second, most likely due to limitations in computational resources caused by the lack of hardware acceleration for machine learning inference.

However, the model made accurate steering predictions, demonstrating a substantial degree of robustness. The results indicate that the frame drops did not hinder the model's performance, as it successfully completed five fully autonomous laps in all four test scenarios, including the three with additional noise.

Nonetheless, it is useful to also numerically quantify the exact impact of the frame drops on the model's behavior. Therefore, post-hoc inference was carried out on a virtual machine by extracting the rosbag messages from the depth camera. This enabled a direct comparison between the real-time inference, which was affected by dropped frames, and the post-hoc inference, in which all frames were fully retained.

Figure 6.17 shows the histogram of the pairwise absolute steering differences under both conditions. The distribution of these differences, when examined for both real-time inference (in which frame drops were experienced) and post-hoc inference (in which all recorded frames were utilized), are extremely similar.

This finding suggests that, even under unintentional frame loss, the model's predictions did not exhibit substantial erratic behavior or significant deviations. The similarity between the two distributions confirms that the frame drops did not meaningfully compromise the model's overall performance.

6.6 Limitations

While additional configurations of the recurrent network could have been explored, this thesis primarily focused on investigating feature extractors. In this context, recurrent networks were employed solely as a component of the pipeline responsible for making sequential predictions. Therefore, further refinements to the recurrent architecture remain beyond the immediate scope of this thesis.

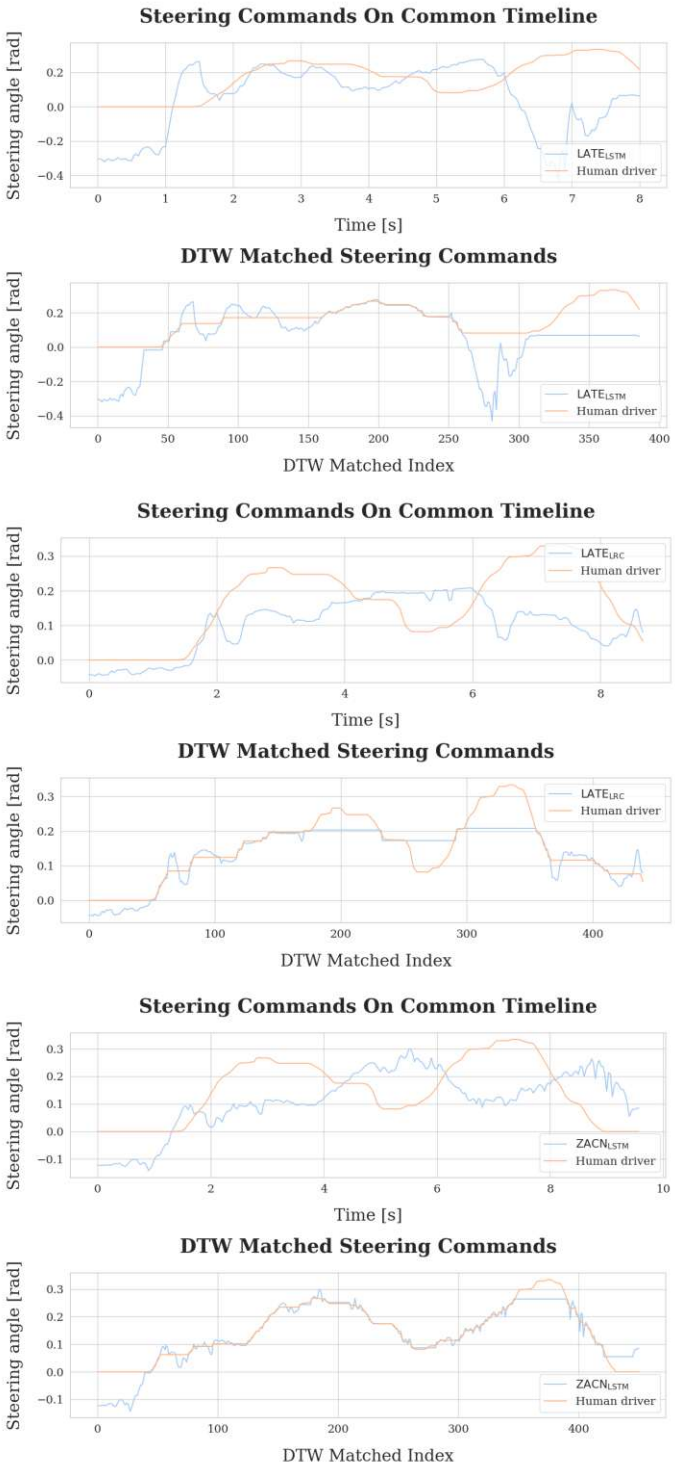


Figure 6.12: Comparison between the steering commands predicted by the models that crashed at position 4 in Figure 6.2 and those of the human expert driver.

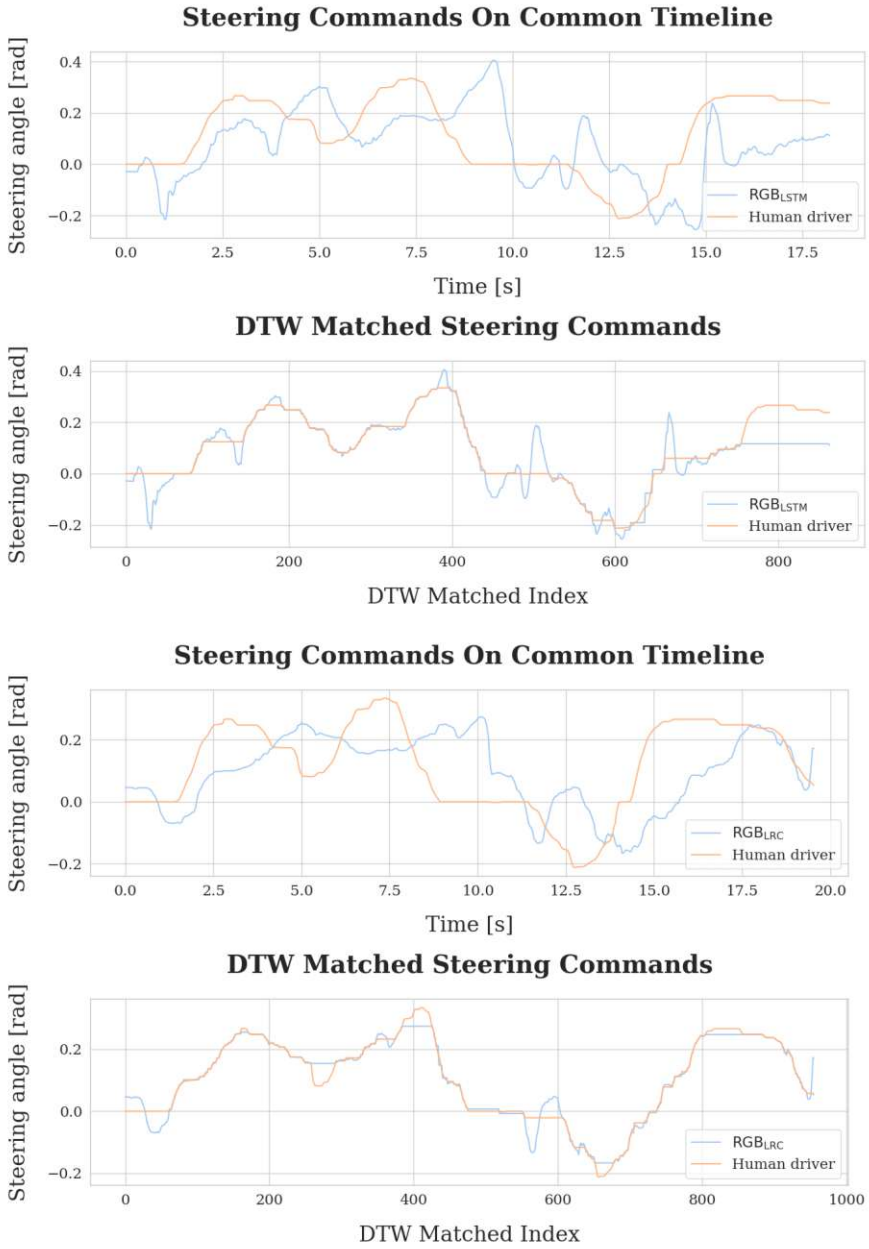


Figure 6.13: Comparison between the steering commands predicted by the RGB_{LSTM} and RGB_{LRC} models and those of the human expert driver, up to the irrecoverable crashes at positions 5 and 6 in Figure 6.2.

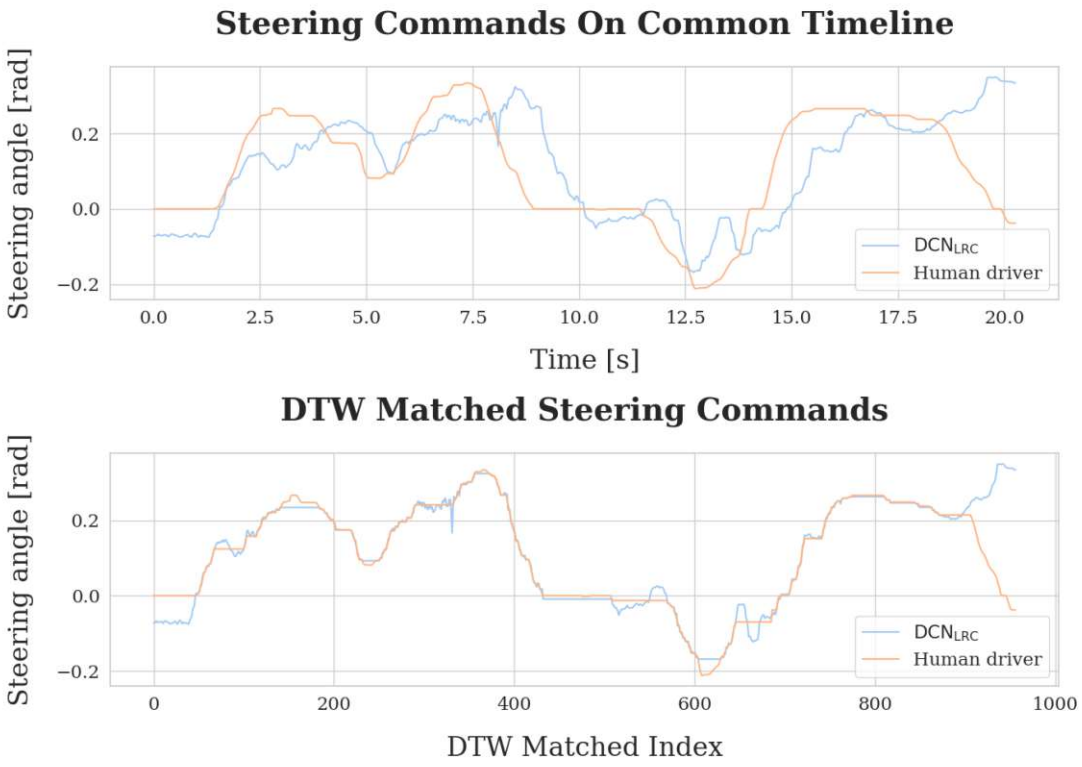


Figure 6.14: Comparison between the steering commands predicted by the DCN_{LRC} model and those of the human expert driver, up to the irrecoverable crash at position 7 in Figure 6.2.

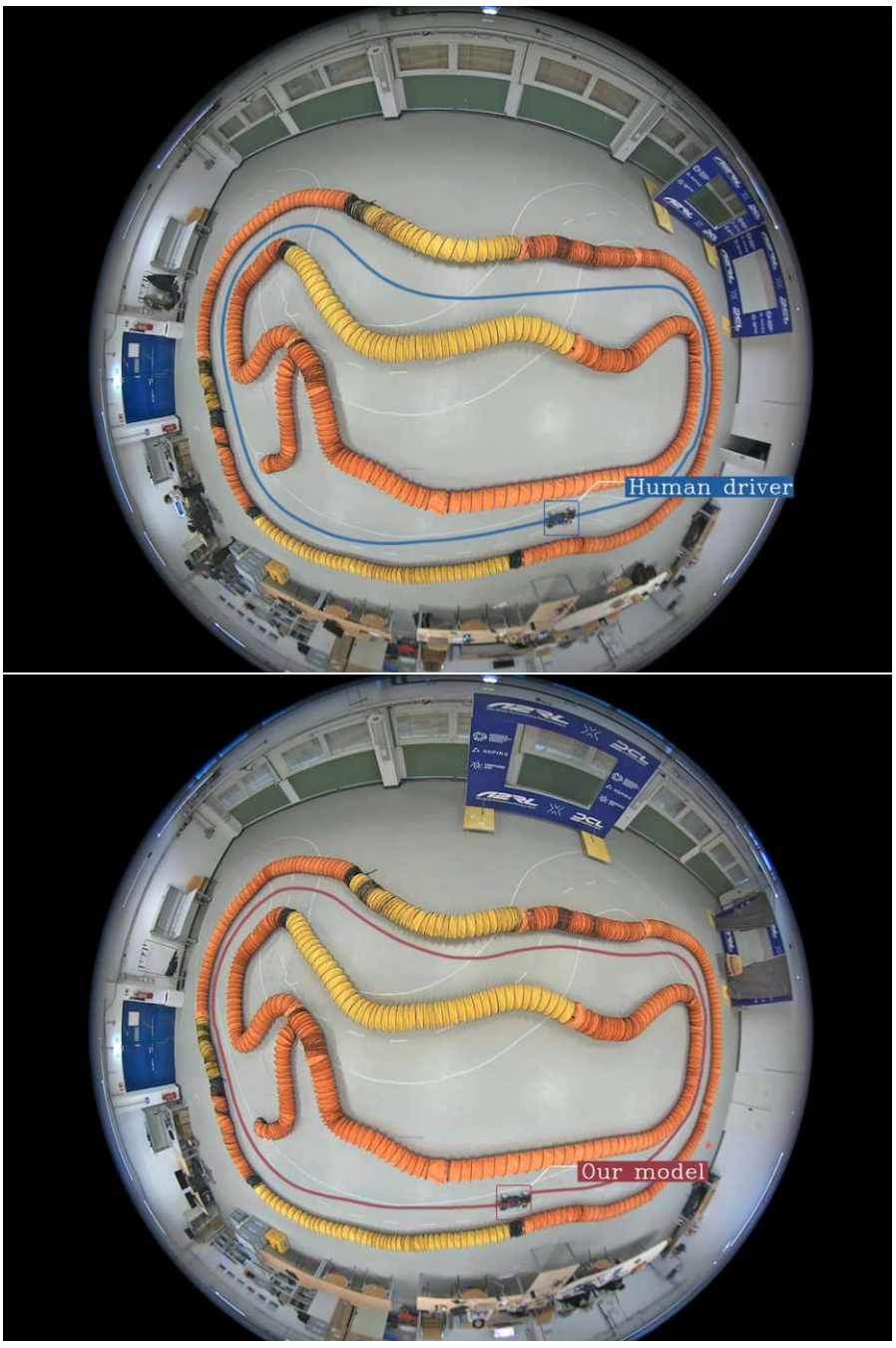


Figure 6.15: The upper image shows the trajectory of the human operator on the test circuit, and the lower image depicts the route followed by the best model, $\text{EARLY}_{\text{LSTM}}$, determined via object tracking. Although the model's trajectory is smooth, it does not entirely replicate the human driving style. From [CFRG25].

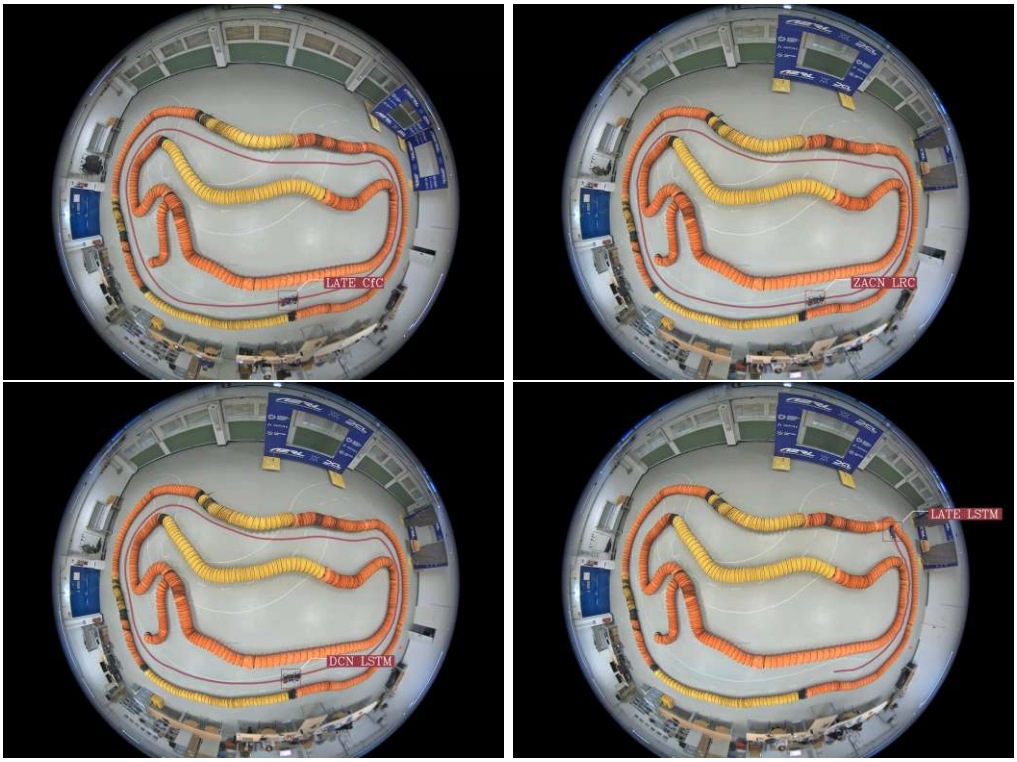


Figure 6.16: From left to right, trajectories of the successful models: LATE_{CfC}, ZACN_{LRC}, DCN_{LSTM}, and the crash of $\text{LATE}_{\text{LSTM}}$, the model that had placed first in Table 6.4.

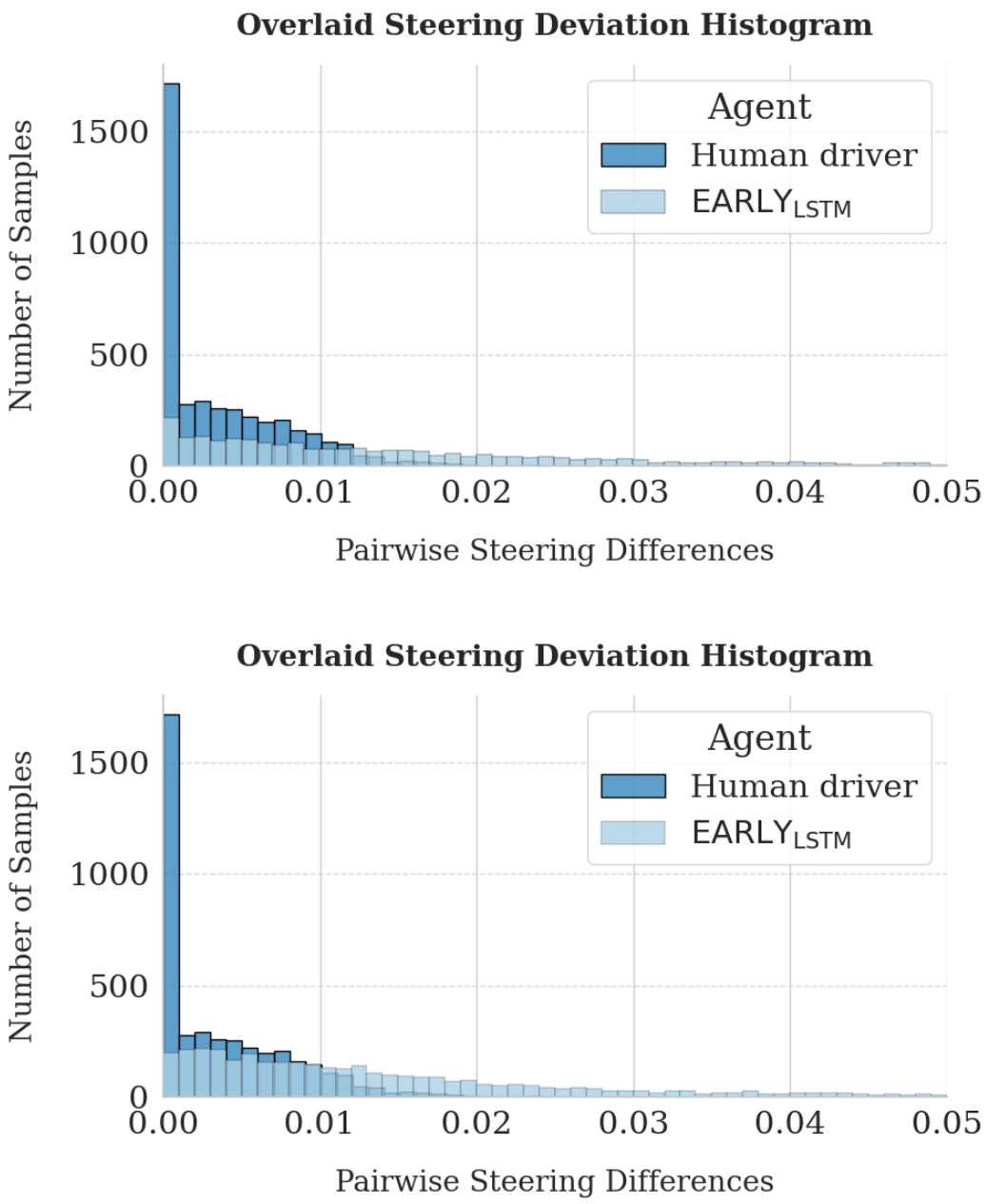


Figure 6.17: Histogram displaying the variations in steering angles between successive predictions, serving as a measure of steering consistency (lower variation denotes smoother control). Distributions of model-generated commands are contrasted with those issued by human operators during closed-loop tests. The top histogram reflects real-time inferences with ≈ 7 frames per second lost due to hardware constraints, while the lower plot was obtained from post-hoc analysis. The close similarity between these distributions indicates that dropped frames had minimal impact on overall performance. From [CFRG25].

CHAPTER 7

Conclusion

Overcoming the sim-to-real gap in the deployment of autonomous agents is an undoubtedly complex challenge, requiring controllers that are not only accurate but also robust. This complexity arises from factors such as noise affecting sensor readings and computational delays from heavy processing, both of which can significantly impair an agent's performance. As human control relies heavily on stereoscopic vision to navigate surroundings, this thesis explores the effects of different RGB-D fusion techniques on autonomous agents.

Based on an in-depth study of state-of-the-art approaches to RGB-D agents, this thesis investigates the potential of multimodal RGB-D recurrent controllers that make accurate and fast predictions even on platforms lacking dedicated hardware acceleration for machine learning inference.

Following the collection of high-quality data and the establishment of the methodology to be used for training the models, the research continues with empirical evaluations. Experimenting in open-loop and closed-loop settings confirms the validity of the systematic testing approach. Through various numerical analysis methods and attention mechanisms, the models' behavior, stability, and resilience to noise are quantified and interpreted.

All in all, this thesis comprises a comprehensive study of developing vision-based neural controllers capable of performing under real-world conditions. Moreover, it paves the way for further advancements in the field, contributing valuable knowledge toward the development of robust autonomous systems.

7.1 Key Insights

To what extent does depth improve steering performance?

First and foremost, this thesis answers the question concerning *the extent to which depth information may improve steering control.*

During closed-loop testing, it is demonstrated that depth perception is not merely advantageous to RGB but rather indispensable. The unimodal RGB feature extractor proves inadequate, crashing in sharp turns during hardware deployment. Models relying solely on RGB information fail to identify the open path in corners and struggle to determine the correct steering direction.

By contrast, the controllers that incorporate RGB-D feature extractors attain full autonomy. As depth information provides crucial spatial awareness, RGB-D agents recognize areas with more free space and adjust their steering accordingly.

Which fusion method provides the best balance between robustness and efficiency?

Secondly, the results reveal that *early depth fusion strikes the best balance between robustness and computational efficiency.*

The early fusion model proves more robust in conditions involving both unpredictable data loss and varying levels of input noise, consistently outperforming late fusion and depth-adaptive fusion models. This finding provides strong evidence in support of early fusion mechanisms as an effective means of integrating multimodal sensory inputs while maintaining computational efficiency.

Can full autonomy be achieved with a lightweight multimodal control system?

Last but not least, this thesis proves *it is possible to achieve full autonomy by using lightweight recurrent controllers with RGB-D input.*

Models using multimodal feature extractors maintain autonomy over continuous laps, with the early fusion extractor overcoming severe sensor noise. Moreover, despite frame drops during real-time inference, the deployed models make consistent steering predictions, confirming the viability of lightweight multimodal control for autonomous agents.

7.2 Further Research

While this thesis proves the importance of multimodal fusion in robust autonomous control, it also opens the way for several research opportunities.

Analyzing attention patterns

On the one hand, a more in-depth examination of attention behavioral patterns within the different recurrent neural networks is planned.

As this thesis suggests that attention plays a crucial role in sequential decision-making, the investigation will continue to implement alternative attention visualization methods beyond VisualBackProp. These new insights could lead to improvements in architectural design, ultimately contributing to safer and more reliable models.

Enhancing the visual field

On the other hand, it is possible to improve the hardware stack and the collected data. More specifically, increasing the agents' field of view is achievable by incorporating multiple cameras in the hardware stack.

Additional cameras positioned on the sides of the agent can enhance perception in challenging scenarios, such as sharp turns or obstacle-rich areas. Clearly, this approach requires revisiting the feature extraction process, as additional input streams need to be processed and fused. However, such an approach has the potential to allow higher driving speeds and more reliable obstacle avoidance.

APPENDIX A

Explanation of Preprint Citation in Figures and Tables

This appendix clarifies that specific figures and tables included in this thesis were first published in the conference paper preprint [CFRG25]. The conference paper, submitted for the 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2025), was prepared as a concise 8-page summary of a subset of the research results obtained during the evolution of the project. The paper was submitted on March 1, 2025, with notification of acceptance to be received on June 30, 2025, and scheduled to be held during October 19-25, 2025.

Given the possibility of the paper being accepted, a preprint was published on arXiv [CFRG25] to facilitate timely citation. The decision to submit the paper to IROS was made because the results were considered relevant to the robotics research community. It is emphasized that the paper is a spin-off of the thesis, summarizing key findings in a condensed format, while the thesis provides a comprehensive account of the research with additional experiments, data, and analyses beyond the scope of the conference paper.

For clarity and proper attribution, each figure and table that has been reused includes an in-caption note stating: "From [CFRG25]". This measure is taken to explicitly indicate the first source of publication of the figures and tables.

The reuse of figures and tables does not compromise the originality or completeness of the thesis content but rather is used to ensure consistency in the communication of results across publications.

APPENDIX B

Steering Angle Histograms

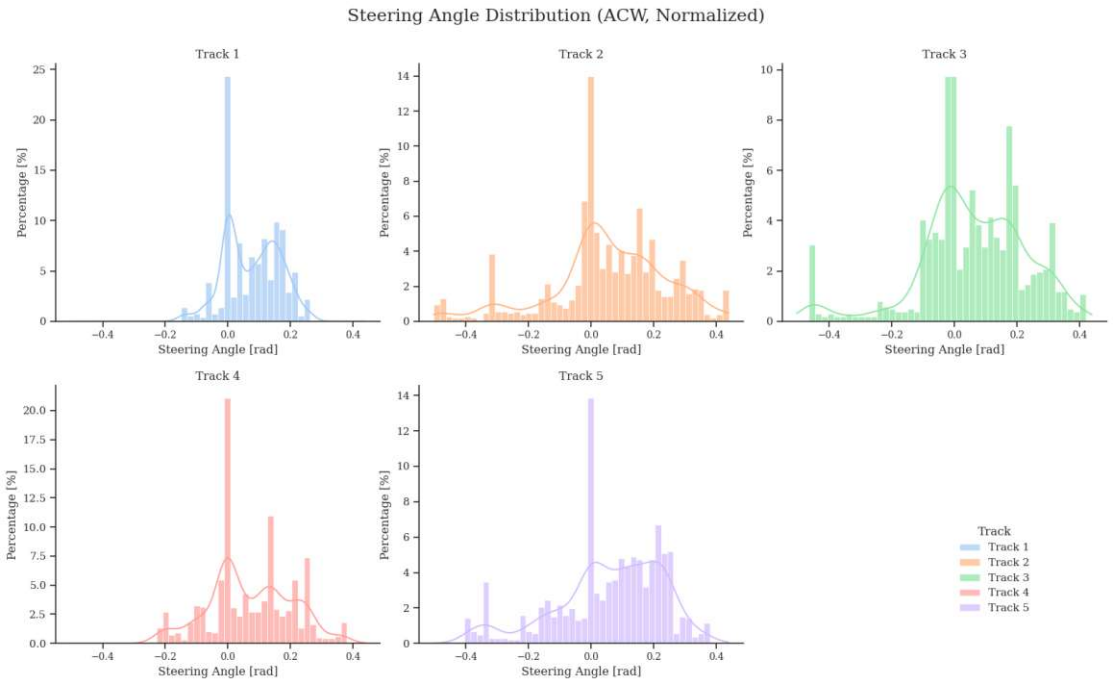


Figure B.1: Steering angle histograms, including KDE curves, generated from 1 lap on each of the tracks in anti-clockwise direction.

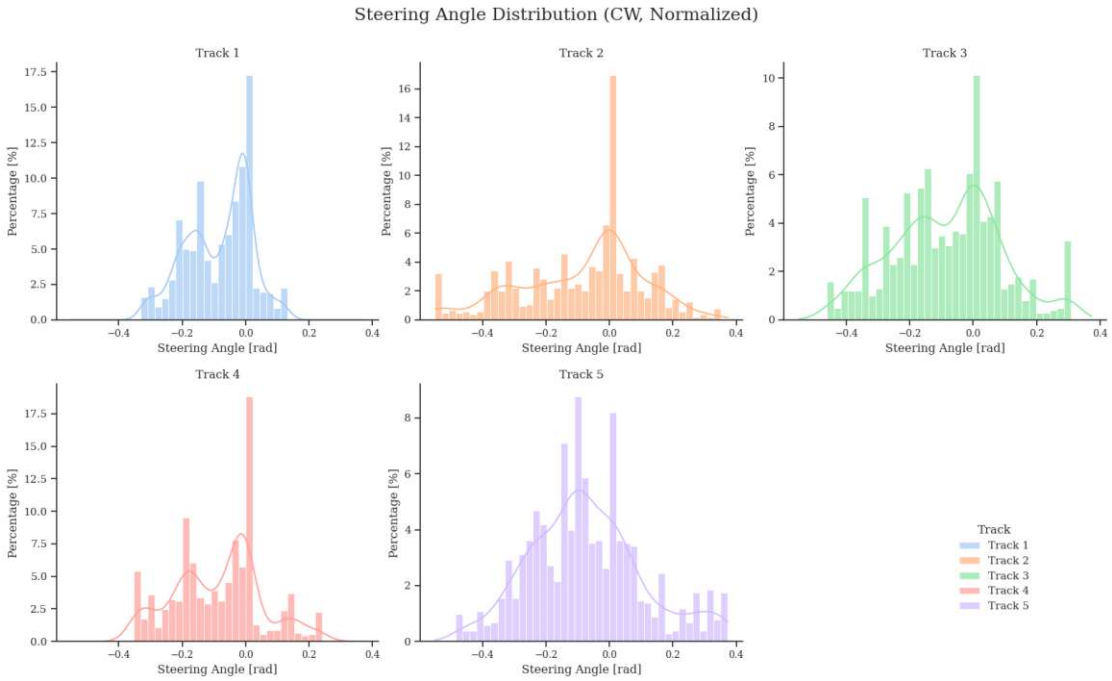


Figure B.2: Steering angle histograms, including KDE curves, generated from 1 lap on each of the tracks in clockwise direction.

List of Figures

List of Tables

Glossary

resource-constrained Here, referring to a platform operating only on the Central Processing Unit (CPU) without dedicated hardware acceleration for machine learning inference.. 1, 26, 55

Acronyms

ACW Anti-clockwise. 29, 45

CfC Closed-form Continuous-time Neural Network. 18, 39, 40, 42, 43, 46, 54, 55

CNN Convolutional Neural Network. 7–11, 13, 21, 22, 36, 37, 75

CPU Central Processing Unit. 26, 81

CW Clockwise. 29

DCN Deformable Convolutional Network. 9, 10, 12, 36

DE Differential Equation. 17, 18

DN Deconvolutional Network. 11, 12

DRL Deep Reinforcement Learning. 23

DSTC Deformably-Scaled Transposed Convolution. 12

DTW Dynamic Time Warping. 51, 52, 57

FC Fully-Connected. 22

GRU Gated Recurrent Units. 19

KDE Kernel Density Estimation. 73, 74, 77, 78

LRC Liquid Resistance Liquid Capacitance Networks. 18, 19, 34, 42, 43, 55, 56

LSTM Long Short-Term Memory. 15, 16, 19, 22, 40, 42, 43, 55, 56, 75

LTC Liquid Time-Constant. 17, 18, 21, 40, 42, 55

MAE Mean Absolute Error. 6

MLP Multi-layer Perceptron. 5, 6, 9, 75

MP McCulloch–Pitts. 3

MSE Mean Squared Error. 6, 38, 42–44, 56, 79

NCP Neural Circuit Policy. 21, 22

ODE Ordinary Differential Equation. 16, 18, 19

RMSE Root Mean Squared Error. 51–54, 56, 57, 80

RNN Recurrent Neural Network. 13–17, 19, 22, 34, 35, 42, 44, 45, 58, 75, 80

ROS Robot Operating System. 27

SSIM Structural Similarity Index. 48–50, 76, 77

Bibliography

[BCC+16] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry D. Jackel, Urs Muller, and Karol Zieba. Visualbackprop: visualizing cnns for autonomous driving. *CoRR*, abs/1611.05418, 2016.

[BDEE23] Hendrik Willem Jordaan Benjamin David Evans and Herman Arnold Engelbrecht. Comparing deep reinforcement learning architectures for autonomous racing. *Machine Learning with Applications*, 14:100496, 2023.

[BMB+16] Alexander Binder, Grégoire Montavon, Sebastian Bach, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. *CoRR*, abs/1604.00825, 2016.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[BRX+22] Stefano B. Blumberg, Daniele Raví, Mou-Cheng Xu, Matteo Figini, Iasonas Kokkinos, and Daniel C. Alexander. Deformably-scaled transposed convolution, 2022.

[BTD+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[BZJ+24] Johannes Betz, Hongrui Zheng, Felix Jahncke, Zirui Zang, Florian Sauerbeck, Y. Rosa Zheng, Joydeep Biswas, Venkat Krovi, and Rahul Mangharam. F1tenth: Enhancing autonomous systems education through hands-on learning and competition. *IEEE Transactions on Intelligent Vehicles*, pages 1–13, 2024.

[CFRG25] Mihaela-Larisa Clement, Mónika Farsang, Felix Resch, and Radu Grosu. Depth matters: Multimodal rgb-d perception for robust autonomous agents, 2025.

[CR74] Edwin Catmull and Raphael Rom. A class of local interpolating splines. In *Computer aided geometric design*, pages 317–326. Elsevier, 1974.

[CRBD18] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *CoRR*, abs/1806.07366, 2018.

[DQX+17] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR*, abs/1703.06211, 2017.

[DTA25] DTAI Research Group. Distance measures for time series (dynamic time warping, fast c implementation), 2025. Accessed: 24-03-2025.

[EBZ+23] Benjamin David Evans, Johannes Betz, Hongrui Zheng, Herman A. Engelbrecht, Rahul Mangharam, and Hendrik W. Jordaan. Bypassing the simulation-to-reality gap: Online reinforcement learning using a supervisor, 2023.

[EJE23] Benjamin D. Evans, Hendrik W. Jordaan, and Herman A. Engelbrecht. Safe reinforcement learning for high-speed autonomous racing. *Cognitive Robotics*, 3:107–126, 2023.

[ESS+15] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin A. Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust RGB-D object recognition. *CoRR*, abs/1507.06821, 2015.

[FNG24] Mónika Farsang, Sophie A. Neubauer, and Radu Grosu. Liquid resistance liquid capacitance networks, 2024.

[Fou25] RoboRacer Foundation. Roboracer - build documentation. `https://roboracer.ai/build`, 2025. Accessed: 20-03-2025.

[FPWW03] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Convolution, 2003. Accessed: 15.02.2025.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[HLA+20] Ramin M. Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. *CoRR*, abs/2006.04439, 2020.

[HLA+22] Ramin Hasani, Mathias Lechner, Alexander Amini, Lucas Liebenwein, Aaron Ray, Max Tschaikowski, Gerald Teschl, and Daniela Rus. Closed-form continuous-time neural networks. *Nature Machine Intelligence*, 4(11):992–1003, November 2022.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[HW18] Isma Hadji and Richard P. Wildes. What do we understand about convolutional networks? *CoRR*, abs/1803.08834, 2018.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[IBM25] IBM. Convolutional neural networks, 2025. Accessed: 10-03-2025.

[Ins24] Inspiros. Torchvision+ deformable convolution networks. `https://github.com/inspiros/tvdcn`, 2024. Accessed: 27-02-2025.

[Int25] Intel. Intel extension for pytorch. `https://github.com/intel/intel-extension-for-pytorch`, 2025. Accessed: 20-03-2025.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[Lan50] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of research of the National Bureau of Standards*, 45(4):255–282, 1950.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBG+23] Michael Loetscher, Nicolas Baumann, Edoardo Ghignone, Andrea Ronco, and Michele Magno. Assessing the robustness of lidar, radar and depth cameras against ill-reflecting surfaces in autonomous vehicles: An experimental study, 2023.

[LHA+20] Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A. Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10):642–652, October 2020.

[MALJ22] Farzeen Munir, Shoaib Azam, Byung-Geun Lee, and Moongu Jeon. Multi-modal fusion for sensorimotor coordination in steering angle prediction, 2022.

[MB89] Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. *Advances in neural information processing systems*, 2, 1989.

[MBB+24] Fouad Makiyeh, Mark Bastourous, Anass Bairouk, Wei Xiao, Mirjana Maras, Tsun-Hsuan Wangb, Marc Blanchon, Ramin Hasani, Patrick Chareyre, and Daniela Rus. Optical flow matters: an empirical comparative study on fusing monocular extracted modalities for better steering, 2024.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[MP69] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 479(480):104, 1969.

[MVLÁCBAM24] Dante Mújica-Vargas, Antonio Luna-Álvarez, Mirna Castro Bello, and Andrés Antonio Arenas Muñiz. *RGB-D Convolutional Recurrent Neural Network to Control Simulated Self-driving Car*, pages 395–416. Springer Nature Switzerland, Cham, 2024.

[PMG24] James Bowman Patrick Mihelich and Ethan Gao. Ros package: cv_bridge. `https://index.ros.org/p/cv{_}bridge/`, 2024. Accessed: 20-03-2025.

[Res23] Felix Resch. Autonomous racing with attention-based neural networks. Master's thesis, Technische Universität Wien, 2023.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[Rob21] Open Robotics. Ros: Home. `https://www.ros.org`, 2021. Accessed: 20-01-2025.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[Sak78] Hiroaki Sakoe. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:159–165, 1978.

[Seg21] Scuderia Segfault. Scuderia segfault. `https://www.tuwien.at/inf/f1tenth/`, 2021. Accessed: 22-03-2025.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SSB14] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Sze22] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[Tav21] Romain Tavenard. An introduction to dynamic time warping. `https://rtavenar.github.io/blog/dtw.html`, 2021.

[Vin68] T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4:52–57, 1968.

[WASD20] Zongwei Wu, Guillaume Allibert, Christophe Stolz, and Cedric Demonceaux. Depth-adapted cnn for rgb-d cameras, 2020.

[WBSS04] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[XCG+19] Yi Xiao, Felipe Codevilla, Akhil Gurram, Onay Urfalioglu, and Antonio M. López. Multimodal end-to-end autonomous driving. *CoRR*, abs/1906.03199, 2019.

[Yan16] Shi Yan. Understanding lstm and its diagrams. `https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714`, 2016. Accessed: 12-03-2025.

[ZGL+20] Yanqiu Zhang, Ruiquan Ge, Lei Lyu, Jinling Zhang, Chen Lyu, and Xiaojuan Yang. A virtual end-to-end learning system for robot navigation based on temporal dependencies. *IEEE Access*, 8:134111–134123, 2020.

[ZHLD18] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results, 2018.

[Zho21] Zhi-Hua Zhou. *Machine learning*. Springer nature, 2021.

[ZKTF10] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, 2010.

[ZWY+24] Mohammed Misbah Zarrar, Qitao Weng, Bakhbyergyen Yerjan, Ahmet Soyyigit, and Heechul Yun. Tinylidarnet: 2d lidar-based end-to-end deep learning model for f1tenth autonomous racing, 2024.