Technische Universität Berlin

Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Temporal Comprehension in Autonomous Drone Racing: Using a Recurrent Convolutional Neural Network that Derives Navigation Decisions from Current and Past Raw Sensor Data

Master Thesis
am Fachgebiet Agententechnologien in betrieblichen Anwendungen und der Telekommunikation (AOT)
Prof. Dr.-Ing. habil. Sahin Albayrak
Fakultät IV Elektrotechnik und Informatik
Technische Universität Berlin

vorgelegt von
Friedrich Clemens Valentin Mangelsdorf

Betreuer:	Dr. rer. nat Yuan Xu,
Gutachter:	Prof. Dr.-Ing. habil. Sahin Albayrak
	Prof. Dr. Odej Kao

Friedrich Clemens Valentin Mangelsdorf
Matrikelnummer: 356686

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift

Abstract

Autonomous navigation methods for drones already achieve remarkable results using CNNs with their keen spatial comprehension of visual sensor data. Humans also rely primarily on their visual perception and spatial comprehension to move through their surroundings. However, unlike CNNs, their spatial comprehension is not limited to what is currently in their field of view, but extends to their memory of what they have already seen. They can link this memory to their often unconscious decisions about how to move to reach a goal. For example, they may incorporate their own motion history or their estimation/anticipation of moving objects into their decisions. So how about trying to achieve this human-like combined, spatial and temporal motion competence in autonomous drone navigation?

Like much research in this area, this thesis draws on autonomous drone racing as a research playground. It takes the agile, high-performance drone racing method of Kaufmann et al. [31] as a baseline and extends the feedforward CNN of the method with a recurrent GRU to incorporate temporal comprehension into the navigation decision-making of the method. The thesis conducts experiments in simulation to evaluate the performance of several feedforward and recurrent variants in the imitation learning process of the method and in drone race tests.

Admittedly, none of the variants investigated in this work comes close to the exceedingly good results of the baseline work, mainly because of much more difficult learning/testing conditions. Yet, the comparison between the recurrent and feedforward variants studied under the same conditions shows that the recurrent variants perform much better in the race test, especially at higher speeds. At the same time, the recurrent variants are more effective in the imitation learning process, which manifests in significantly fewer rollouts and aggregated training samples, but comes at the price of significantly longer training times.

Zusammenfassung

Autonome Navigationsmethoden für Drohnen erzielen bereits bemerkenswerte Ergebnisse, indem sie CNNs mit ihrem scharfen räumlichen Verständnis auf visuelle Sensordaten anwenden. Auch Menschen verlassen sich in erster Linie auf ihre visuelle Wahrnehmung und ihr räumliches Vorstellungsvermögen, um sich in einer Umgebung zu bewegen. Im Gegensatz zu CNNs ist ihr räumliches Verständnis jedoch nicht auf das beschränkt, was sich gerade in ihrem Blickfeld befindet, sondern erstreckt sich auch auf ihre Erinnerung an das, was sie bereits gesehen haben. Sie können diese Erinnerung mit ihren oft unbewussten aber zielführenden Bewegungsentscheidungen verknüpfen. So können sie beispielsweise ihre eigene Bewegungsgeschichte oder ihre Einschätzung/Antizipation von sich bewegenden Objekten in ihre Entscheidungen einbeziehen. Wie wäre es also, diese dem Menschen ähnliche kombinierte räumliche und zeitliche Bewegungskompetenz bei der autonomen Navigation von Drohnen zu verwenden?

Wie viele andere Arbeiten in diesem Bereich nimmt auch diese Arbeit autonomes Drohnenrennen als Forschungsspielfeld. Sie nimmt die agile, hochleistungsfähige Methode für Drohnenrennen von Kaufmann et al. [31] als Grundlage und erweitert das vorwärtsgerichtete CNN der Methode mit einer rekurrenten GRU, um zeitliches Verständnis in die Navigationsentscheidungen der Methode einzubeziehen. In dieser Arbeit werden Simulationsexperimente durchgeführt, um die Leistung mehrerer vorwärtsgerichteter und rekurrenter Varianten im Imitationslernprozess der Methode und in Testrennen zu bewerten.

Zwar kommt keine der hier untersuchten Varianten an die außerordentlich guten Ergebnisse der Grundlagenarbeit heran, was vor allem an den wesentlich schwierigeren Lern-/Testbedingungen liegt. Der Vergleich zwischen den unter gleichen Bedingungen untersuchten rekurrenten und vorwärtsgerichteten Varianten zeigt jedoch, dass die rekurrenten Varianten im Renntest deutlich besser abschneiden, insbesondere bei höheren Geschwindigkeiten. Gleichzeitig sind die rekurrenten Varianten beim Imitationslernen effektiver, was sich in deutlich weniger Ausrollvorgängen und aggregierten Trainingsbeispielen äußert, aber mit deutlich längeren Trainingszeiten erkauft wird.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Advances in drone technology in recent years have opened the door to many civilian application possibilities, especially in the commercial sector [13]. Today, drones are already standard in aerial inspection services [12, 1, 3]. In the near future, the areas of infrastructure, transport, insurance, media and entertainment, telecommunication, agriculture, safety and mining are particularly promising for drones [38]. In contrast, genuine breakthroughs of more visionary applications, such as drone delivery or drone taxis, are, if at all, only expected in the more distant future [50]. Reasons for this are strict regulations, skeptical public attitudes and technical problems [50]. Regarding the latter, autonomous navigation methods are not yet robust enough for the reliable deployment in open-world environments of high uncertainty such as densely populated urban areas [7]. This makes the autonomous navigation of drones of great importance in current research. Loquercio and Scaramuzza [37] categorize existing autonomous navigation methods for drones into classical methods and modern, machine learning based methods.

Classical autonomous navigation methods follow the scheme of mapping-localization-planning-tracking. Simpler approaches track pre-programmed waypoints based on GNSS and are thus limited to flight environments that first, have a reliable GNSS signal (which often is not the case for urban areas, mountains, indoor sites, caves, etc.) and second, are free of obstacles, since functions to cope with them are absent. More complex approaches use SLAM algorithms to generate a map of the flight environment and localize in it based on the drone's visual or range sensors [44]. Path planning algorithms (e.g., [5], [11]) apply to generate collision-free trajectories through the generated maps, which are tracked while estimating the drone state based on data from the drone's inertial measurement unit (IMU) and the output of SLAM algorithms (e.g., [35], [55], [53], [36]). Theoretically, these approaches are deployable in uncon-

trolled environments with unknown obstacles and other agents. Practically, the mapping coerces global consistency, which likely makes computations too complex for real-time execution onboard drones, especially for non-static environments.

Modern autonomous navigation methods use machine learning to replace the classical mapping-localization (or even the planning and tracking for more end-to-end methods) with the perception of the drone's onboard sensors (e.g., camera, range, IMU) and the decision-making of an ANN. The way of learning further subdivides these methods into reinforcement learning (RL) and imitation learning (IL) based methods. In RL, the ANN learns with "trial-and-error" [54] from direct environmental interactions and thus avoids "distributional mismatch" [2] of training vs. testing, which is typical in IL. However, this comes with the expense of a substantially higher sample complexity compared to IL [63]. This disadvantage is severe for drones because their limited flight endurance makes the learning process inefficient and collisions are uncontrolled and hence costly and dangerous [54].

A popular research playground is autonomous drone racing (e.g., [42], [27], [58]) because of the controlled flight environment, the straightforward definition of goals and the measurability and comparability of performance. Methods for autonomous drone racing usually integrate feedforward convolutional neural networks (CNN) to map the current color or depth image to action (e.g., [49], [30], [28]). CNNs already achieve a high, spatial comprehension of how the drone's sensors perceive the immediate environment. However, this alone may not be enough for the long-term objective of safely applying autonomous drones in open-world environments. As humans are able to navigate safely there, it seems promising to learn human-like abilities. Temporal comprehension, besides spatial comprehension, has an important function in human navigation. For example, after entering a room through a door, a person can leave the room knowing that the door is behind him and does not need to keep the door in sight to do so. Further, a person requires observing a sequence of images to estimate the velocity and motion direction of herself or himself, obstacles or other agents. In machine learning, recurrent neural networks can establish temporal connections. It is interesting to research the effect of using recurrent neural networks in autonomous drone racing.

1.2 Approach and Goals

This thesis aims to investigate whether temporal comprehension induced by a recurrent neural network can be beneficial for autonomous navigation in drone racing. To do this, the thesis uses the vision-based autonomous navigation method proposed by Kaufmann et al. [31] as a baseline. In the tests conducted by the authors, the baseline method stood out from the compared methods with high reliability and agility along dynamic flight curves through the racetrack. The baseline method has a hybrid structure: an artificial neural network inferring navigation decisions from the images of the drone's onboard

camera and a planning-control stack translating these decisions into flight movement. The baseline ANN is a serial connection of a convolutional neural network (CNN) and fully connected (FC) layers. The CNN extracts visual features from imagery, enabling the baseline method to comprehend spatially what is in view of the drone's onboard camera.

This thesis extends the baseline ANN with a multi-layer gated recurrent unit (GRU). Theoretically, this recurrent neural network variant enables the autonomous navigation method to remember and to establish temporal connections. Specifically, this means that navigation decisions base on both current and past sensor data. Considering that, first, navigation can be seen as sequentially making decisions regarding how to move through space, and second, the thereby resulting trajectories are 4D objects in space and time, the CNN-GRU approach of this thesis, by incorporating both spatial and temporal understanding, likely improves race performance. This thesis investigates this hypothesis in simulated drone races. Different variants of ANNs are tested for their performance at different maximum speeds and for their ability to cope with intermediate target loss. The latter refers to the event that the next race gate leaves the FOV of the drone's camera (e.g., because of a sharp turn between two race gates) and the drone can only make meaningful navigation decisions based on the memory induced by the GRU.

1.3 Structure of the Thesis

Chapter 2 provides background information on the thesis topic, including imitation learning with dataset aggregation and the gated recurrent unit architecture. Chapter 3 presents the baseline autonomous navigation method comprising the ANN module, the planning module, the control stack and the expert system. Special attention is paid to the ANN module, which extends the baseline ANN with the temporal comprehension of the GRU. Chapter 4 presents the design of the experiments conducted in this thesis, including the simulation setup, the imitation learning process and the race tests. Chapter 5 evaluates the experimental results focussing on the impact of temporal comprehension on performance. Chapter 6 concludes this thesis. Chapter 6.3 provides additional related information.

Chapter 2

Background

2.1 Imitation Learning with Dataset Aggregation

This section first, defines the general imitation learning problem, second, presents behavioral cloning as the simplest form of direct policy learning and third, introduces the more sophisticated direct policy learning method of dataset aggregation, which was originally proposed as DAgger by Ross et al. [51]. The experiments of this thesis (see chapter 3) dataset aggregation to the imitation learning problem of the ANN module in the autonomous navigation method (see chapter 4).

Imitation learning is an area of machine learning that considers the task of learning to imitate (i.e., behave like) an expert in a sequential decision-making problem [62]. It applies naturally to learning problems where it is easier to demonstrate the desired behavior of how to attain a goal than to define intermediate goals and rewards (as in supervised and reinforcement learning, respectively) that lead to the desired behavior [45]. In terms of supervision, imitation learning is therefore between supervised learning and reinforcement learning. Imitation learning performs well in tasks related to human intent or behavior, e.g., pedestrian avoidance [64], speech animation [59], sport analytics [32] and gaming [60]. Regarding planning and control in autonomous navigation, imitation learning has become state-of-the-art [40]. According to their approach, imitation learning methods divide into direct policy learning and inverse reinforcement learning [2]. Direct policy learning optimizes a policy class to approximate the expert's behavior by reducing the imitation learning problem to a single or a sequence of supervised learning problems. Inverse reinforcement learning optimizes a reward function to approximate the expert's underlying intent and deploys reinforcement learning methods to optimize a policy class to maximize the identified reward.

Problem Definition
An imitation learning problem includes: a system, an expert, a policy class, a loss

function, and a learning algorithm [62]. The goal is to find the policy within the policy class that approximates the expert's decision-making on the system's actions as closely as possible.

The system interacts with its real or simulated environment. At time step t, the system is in the state $\underline{x}_t \in \mathcal{X}$ and executes the action $\underline{u}_t \in \mathcal{U}$ where $\mathcal{X}$ and $\mathcal{U}$ are the spaces of all existing system states and actions, respectively. The system is considered a Markov decision process whose initial state is randomly distributed [2]. Thus, a probabilistic transition model can describe the system dynamics providing the conditional probability distribution over the system state given the system state and action from the previous time step

$$p\left(\underline{x}_t | \underline{x}_{t-1}, \underline{u}_{t-1}\right). \tag{2.1}$$

The expert is a human or a computer program that makes state-based action decisions, which cause the system to act as desired. The expert policy, which maps system states to expert actions, represents the expert's decision-making process

$$\pi^* : \mathcal{X} \to \mathcal{U}; \quad \underline{x}_t \mapsto \underline{u}_t^*. \tag{2.2}$$

Applying the expert policy on a system state yields a demonstration, i.e., a state-action pair $(\underline{x}_t, \underline{u}_t^*)$.

The policy class defines the search space for finding the optimal policy, i.e., the policy that most closely resembles the expert policy. Usually, the policy class is an ANN

$$\pi_{\underline{\theta}} : \mathcal{X} \to \mathcal{U}; \quad \underline{x}_t \mapsto \pi_{\underline{\theta}}\left(\underline{x}_t\right) \tag{2.3}$$

where the vector $\underline{\theta}$ contains the trainable parameters of the ANN.

Both, the expert policy and members of the policy class, can roll out. The rollout of a general policy $\pi : \mathcal{X} \to \mathcal{U}$ denotes the process, in which the system, starting from the initial state $\underline{x}_0 \sim p\left(\underline{x}_0\right)$, interacts with the environment based on the actions inferred by that policy for a certain number of time steps

$$p\left(\underline{x}_t | \underline{x}_{t-1}, \pi\left(\underline{x}_{t-1}\right)\right), \; t \in \{0, \ldots, T\}. \tag{2.4}$$

A rollout produces a trajectory, which is the sequence of the state-action pairs visited during rollout

$$\tau = \left(\underline{x}_t, \pi\left(\underline{x}_t\right)\right)_{t \in \{0, \ldots, T\}}. \tag{2.5}$$

The policy induces the state distribution of a rollout, i.e., the probability that the system visits a state, which would therewith become part of the rollout trajectory. The rollout state distribution is the average of the state distributions over the time steps of the rollout.

$$p\left(\underline{x} | \pi\right) = \frac{1}{T+1} \sum_{t=0}^{T} p\left(\underline{x}_t | \pi\right). \tag{2.6}$$

The loss function measures how much an action predicted by a policy for a system state deviates from the action demonstrated by the expert for the same state

$$L_\pi : \mathcal{X} \to \mathbb{R}; \quad \underline{x}_t \mapsto L_\pi \left(\pi \left(\underline{x}_t \right), \pi^* \left(\underline{x}_t \right) \right). \tag{2.7}$$

Applied on a set of demonstrations, the loss function quantifies how closely a policy imitates the expert policy.

The learning algorithm searches for the optimal policy $\hat{\pi}^*$ within the policy class by minimizing the total imitation loss, whereby the state distribution of the loss evaluation is equal to the distribution when rolling out the optimal policy. As a result, the state distributions at learning and test rollouts do not differ, which ensures that low training losses transfer to high test performances. In machine learning, this corresponds to the training of the ANN, in which the algorithm updates the trainable parameters of the ANN in accordance to the evaluated loss

$$\hat{\pi}^* = \underset{\underline{\theta}}{\operatorname{argmin}} \sum_{\underline{x} \sim p\left(\underline{x} | \pi_{\underline{\theta}}\right)} L_{\pi_{\underline{\theta}}} \left(\underline{x} \right). \tag{2.8}$$

The above optimization is a "non-i.i.d. supervised learning problem" [51]. As the system dynamics are unknown or too complex, training data can only be collected in rollouts. Thereby, the distribution of the training data is induced by the rolled out policy and hence neither independent and identically distributed (i.i.d.) across the system state space $\mathcal{X}$ nor equal to the distribution induced by the optimal policy yet unknown. Because of the interdependence of policies and training data distributions, closing the "distributional mismatch" [2] is a non-convex optimization problem, which is significantly more difficult and time-consuming than common supervised learning.

Behavioral Cloning

Behavioral cloning reduces the imitation learning problem to a single, convex supervised learning problem on a pre-collected training dataset of expert trajectories, where each trajectory is the product of an expert policy rollout and contains the thereby demonstrated state-action pairs

$$D = \left\{ \tau_i^* \right\}_{i \in \{1,2,\ldots\}}, \quad \tau_i^* = \left(\underline{x}_{i,t}, \underline{u}_{i,t}^* \right)_{t \in \{0,1,\ldots\}}. \tag{2.9}$$

The optimization problem of behavioral cloning is to minimize the total imitation loss on the demonstrations $(\underline{x}_t, \underline{u}_t^*)$ in the trajectories τ_i of the pre-collected dataset D

$$\begin{aligned} \hat{\pi}^* &= \underset{\underline{\theta}}{\operatorname{argmin}} \sum_{\tau_i \in D} \sum_{(\underline{x}_t, \underline{u}_t^*) \in \tau_i} L \left(\pi_{\underline{\theta}} \left(\underline{x}_t \right), \underline{u}_t^* \right) \\ &= \underset{\underline{\theta}}{\operatorname{argmin}} \sum_{\underline{x} \sim p(\underline{x} | \pi^*)} L_{\pi_{\underline{\theta}}} \left(\underline{x} \right), \end{aligned} \tag{2.10}$$

given the expert rollout state distribution

$$p\left(\underline{x}|\pi^*\right) = \frac{1}{T+1}\sum_{t=0}^{T} p\left(\underline{x}_t|\pi^*\right). \tag{2.11}$$

Compared to general imitation learning, behavioral cloning generally exhibits different distributions at training and testing

$$p\left(\underline{x}|\pi^*\right) \neq p\left(\underline{x}|\pi_{\underline{\theta}}\right). \tag{2.12}$$

This plus the fact that the training data is generally not i.i.d. across the state space makes it likely that the ANN, when rolling out, faces states not seen during training. As a result, low training losses do not guarantee high test performances. This is especially true for a longer rollouts because the upper bound of the error grows quadratically with the number of time steps T [62]. Although the above optimization minimizes the one-step loss of the ANN along the expert trajectories, when the ANN rolls out, these one-step errors accumulate along the ANN trajectory and will very likely cause non-negligible deviations from the expert trajectories of the training dataset.

Dataset Aggregation
Dataset aggregation (originally proposed as DAgger [51]) extends behavioral cloning iteratively. It reduces the imitation learning problem to a sequence of convex supervised learning problems. The fundamental concept of the method takes the following steps.

1. Roll out the expert policy π^*, record the thereby resulting trajectory and initialize the aggregated dataset with that trajectory $D_0 = \tau_0$. Train the ANN with supervised learning on the aggregated dataset to receive $\pi_{\underline{\theta},0}$.

2. Until the ANN exhibits the desired behavior when rolling out, do for iteration $i = 1, 2, \ldots$

   (a) Rollout the ANN from the last training $\pi_{\underline{\theta},i-1}$ and record the thereby visited states $\left(\underline{x}_{i,t}\right)_{t\in\{0,\ldots,T_i\}}$.

   (b) Apply the expert policy to the recorded states to receive a trajectory of demonstrations $\tau_i = \left(\underline{x}_{i,t}, \pi^*\left(\underline{x}_{i,t}\right)\right)_{t\in\{0,\ldots,T_i\}}$.

   (c) Add the labeled trajectory to the aggregated dataset $D_i = D_{i-1} \cup \tau_i$.

   (d) Train the ANN with supervised learning on the aggregated dataset to receive $\pi_{\underline{\theta},i}$.

In contrast to behavioral cloning, dataset aggregation requires first, access to the system for executing ANN rollouts and second, an interactive expert providing feedback

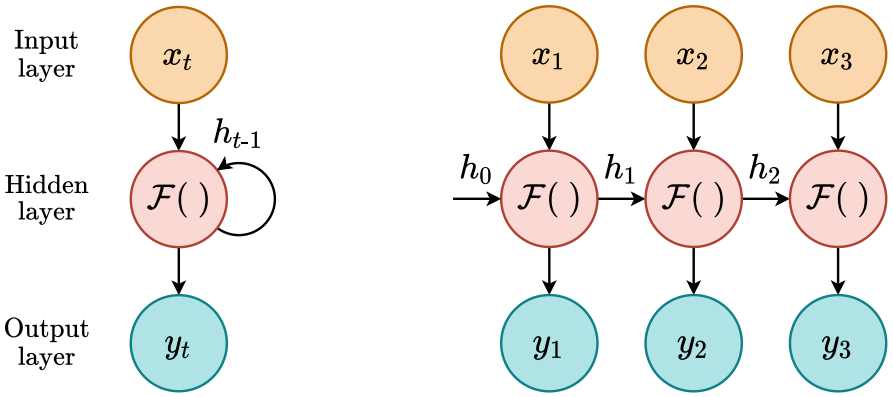
on the thereby visited states [62]. In the iterative learning process, dataset aggregation remembers and confronts the ANN with its mistakes made in rollouts corrected by the expert. Ideally, this procedure reduces the state distribution differences between learning and rollouts to the point where the trainable parameters of the ANN converge to their optimal values for the general imitation learning problem.

2.2 Gated Recurrent Unit

In this thesis, the ANN module of the autonomous navigation method (see section 3) is extended with the gated recurrent unit (GRU), which is a special recurrent neural network (RNN) architecture proposed by Cho et al. [9], with the aim to involve temporal comprehension in the navigation decision-making. This section introduces the class of RNNs and relates the GRU to standard RNNs and the more prevalent long short-term memory (LSTM). Moreover, it provides information regarding first, the state and gating mechanisms that make the GRU able to comprehend temporally at inference and second, how the GRU learns with backpropagation through time.

Recurrent Neural Networks
RNNs contrasts with classical feedforward ANNs, which forward information only towards subsequent layers, by featuring dynamic properties that stem from implementing feedback connections [24] (see fig. 2.1a). As previously inferred states re-enter the network, the output of an RNN depends not only on the current but also on prior inputs. In this sense, RNNs are qualified to process and infer from entire sequences of data points. In the case of time-series data, this equates to temporal comprehension and memory [25]. RNNs train on sequential data with backpropagation through time (BPTT) [47] which is basically the application of standard backpropagation (e.g., [48]) on the unfolded representation of an RNN (see fig. 2.1b). This representation exhibits a layer for each data point in the processed input sequence and is construable as a feedforward ANN that shares its trainable parameters across its layers. The longer the input sequences, the deeper the unfolded representation of an RNN becomes. The training of RNNs on long input sequences is hence prone to the vanishing gradient problem [22], which slows down or ceases the convergence of the trainable parameters of the RNN. As a result, standard RNNs have difficulties learning to connect to information deep in the past [4].

In 1997, Hochreiter and Schmidhuber [23] introduced the long short-term memory (LSTM), which is today's predominant RNN architecture [56]. A standard LSTM layer recurrently maintains a cell state and a hidden state. This means that, besides the current data point of the input sequence, the layer re-inputs the fed back cell state and hidden state from the previous sequential step. Furthermore, the LSTM layer implements three



(a) Folded (b) Unfolded

Figure 2.1: Folded representation of an RNN with a single hidden layer inferring $y_t = \mathcal{F}(x_t, h_{t-1})$ from the current input and the previous hidden state and the time-unfolded representation when processing the input sequence $(x_i)_{i\in\{1,2,3\}}$.

gating mechanisms to control the information flow within the layer. A gating mechanism is basically the Hadamard product of a state and a gate, which is a vector whose entries are within the interval from zero to one. Consequently, a gate applied to a state controls the flow of the state elements within the range from zero to full flow. The forget gate of the LSTM layer controls the flow from the previous to the current cell state. The input gate controls the flow from the previous hidden state and the current data point of the input sequence to the current cell state. And the output gate controls the flow from the current cell state to the current hidden state. By this design of recurrent states with gated information flow, the training of the LSTM is essentially robust to the vanishing gradient problem [47]. As a result, the LSTM is, compared to standard RNNs, more capable of learning to remember long-term dependencies within input sequences.

The GRU, which was proposed 17 years after the LSTM by Cho et al. [9], is a lighter version of the LSTM. It refrains from the cell state and therewith the output gate of the LSTM and only has a hidden state and two gating mechanisms. Therewith, the GRU has less trainable parameters than the LSTM and is less memory efficient. Nevertheless, it preserves the robustness towards the vanishing gradient problem [25]. Various empirical studies compared the GRU and the LSTM. Greff et al. [18] found no substantial performance differences between the GRU and several LSTM variants in speech and handwritten text recognition and music representation. Chung et al. [10] detected an equally well performance of the GRU and the LSTM processing raw speech and polyphonic music data. In the comparative study of Yin et al. [61], the GRU slightly outperforms the LSTM in five of seven natural language processing tasks. Kaiser and Sutskever [29] achieved better results with a GRU-based than with an LSTM-based net-

work in algorithm learning. Considering the above findings and the lower complexity, this thesis uses the GRU and not the LSTM for the ANN module of the autonomous navigation method.

Inference

The following presents the mathematics of the inference of a single GRU layer in accordance with the PyTorch implementation[1]. This includes the layer's two gating mechanisms and its computations of the candidate state and the hidden state.

Without loss of generality, let the sequential data to be processed by the GRU layer be a batch of time series of data points

$$\left(\underline{x}_t \in \mathbb{R}^{N^{\text{in}}}\right)_{t\in\{1,\dots,N^{\text{seq}}\},i}, \quad i \in \left\{1,\dots,N^{\text{batch}}\right\}, \tag{2.13}$$

where N^{batch} is the batch size, N^{seq} is the sequence length and N^{in} is the dimensionality of the data points. Let the initial batch of hidden states be

$$\underline{h}_{0,i} \in [-1,1]^{N^{\text{hidden}}}, \quad i \in \left\{1,\dots,N^{\text{batch}}\right\}, \tag{2.14}$$

where N^{hidden} is the dimensionality of the hidden state, which is a design parameter of the GRU layer. The values of the initial hidden states $\underline{h}_{0,i}$ are typically zero or noisy [65] but can also be learned (e.g., [16]). The GRU layer processes the time series included in the batch parallelly and the data points of the individual time series successively. At the current time step t, the layer's input comprises the batch of the current data points and the fed back batch of previously outputted, hidden states

$$\left(\underline{x}_{t,i},\ \underline{h}_{t-1,i}\right), \quad i \in \left\{1,\dots,N^{\text{batch}}\right\}. \tag{2.15}$$

For simplification, the following text only refers to the parallelly computed, individual elements of the processed batch. However, the following equations yet explicitly refer to the entire batch.

At every incoming input, the GRU layer initially computes its two gates. The current reset gate

$$\underline{g}^{\text{r}}_{t,i} = \mathcal{F}^{\text{r}}\left(\underline{x}_{t,i}, \underline{h}_{t-1,i}\right), \quad i \in \left\{1,\dots,N^{\text{batch}}\right\} \tag{2.16}$$

results from the map

$$\begin{aligned}\mathcal{F}^{\text{r}} : \left(\mathbb{R}^{N^{\text{in}}},\ [-1,1]^{N^{\text{hidden}}}\right) &\to [0,1]^{N^{\text{hidden}}} \\ (\underline{x}, \underline{h}) &\mapsto \overset{\odot}{\sigma}\left(\underline{\underline{A}}^{\text{r}}_{x}\underline{x} + \underline{b}^{\text{r}}_{x} + \underline{\underline{A}}^{\text{r}}_{h}\underline{h} + \underline{b}^{\text{r}}_{h}\right).\end{aligned} \tag{2.17}$$

[1] `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`, accessed on October 5, 2022

The above map to the reset gate comprises two steps. First, it linearly transforms the current data point and the previous hidden state with the trainable weight matrices and bias vectors

$$\begin{aligned} \underline{\underline{A}}_x^{\mathrm{r}} &\in \mathbb{R}^{N^{\mathrm{hidden}} \times N^{\mathrm{in}}}, & \underline{b}_x^{\mathrm{r}} &\in \mathbb{R}^{N^{\mathrm{hidden}}}, \\ \underline{\underline{A}}_h^{\mathrm{r}} &\in \mathbb{R}^{N^{\mathrm{hidden}} \times N^{\mathrm{hidden}}}, & \underline{b}_h^{\mathrm{r}} &\in \mathbb{R}^{N^{\mathrm{hidden}}}, \end{aligned} \tag{2.18}$$

whereby the user has the design option to disable all biases of the GRU layer. This is tantamount to set the above and the below biases of the layer to zero and consider them not trainable. Second, the standard sigmoid function [19] (see fig. 2.2)

$$\sigma : \mathbb{R} \to [0,1] \,;\, x \mapsto \frac{1}{1+e^{-x}} \tag{2.19}$$

applies element-wise (denoted with the accent $\odot$) to the sum of these two linear transformations. The sigmoid function limits the values of the reset gate to the interval between zero and one. This is characteristic of a gating mechanism, which targets at only damping, not amplifying or reversing the individual values of the state.

The current update gate

$$\underline{g}_{t,i}^{\mathrm{u}} = \mathcal{F}^{\mathrm{u}}\left(\underline{x}_{t,i}, \underline{h}_{t-1,i}\right), \quad i \in \left\{1, \ldots, N^{\mathrm{batch}}\right\} \tag{2.20}$$

results from the map

$$\begin{aligned} \mathcal{F}^{\mathrm{u}} : \left(\mathbb{R}^{N^{\mathrm{in}}}, \, [-1,1]^{N^{\mathrm{hidden}}}\right) &\to [0,1]^{N^{\mathrm{hidden}}} \\ (\underline{x}, \underline{h}) &\mapsto \overset{\odot}{\sigma}\left(\underline{\underline{A}}_x^{\mathrm{u}} \underline{x} + \underline{b}_x^{\mathrm{u}} + \underline{\underline{A}}_h^{\mathrm{u}} \underline{h} + \underline{b}_h^{\mathrm{u}}\right). \end{aligned} \tag{2.21}$$

The above map to the update gate is the same as the map to the reset gate but has its own trainable weight matrices and bias vectors

$$\begin{aligned} \underline{\underline{A}}_x^{\mathrm{u}} &\in \mathbb{R}^{N^{\mathrm{hidden}} \times N^{\mathrm{in}}}, & \underline{b}_x^{\mathrm{u}} &\in \mathbb{R}^{N^{\mathrm{hidden}}}, \\ \underline{\underline{A}}_h^{\mathrm{u}} &\in \mathbb{R}^{N^{\mathrm{hidden}} \times N^{\mathrm{hidden}}}, & \underline{b}_h^{\mathrm{u}} &\in \mathbb{R}^{N^{\mathrm{hidden}}}. \end{aligned} \tag{2.22}$$

Knowing the current reset gate, the GRU layer computes the candidate state

$$h_{t,i}^{\mathrm{c}} = \mathcal{F}^{\mathrm{c}}\left(\underline{x}_{t,i}, \underline{h}_{t-1,i}\right), \quad i \in \left\{1, \ldots, N^{\mathrm{batch}}\right\} \tag{2.23}$$

with the map

$$\begin{aligned} \mathcal{F}^{\mathrm{c}} : \left(\mathbb{R}^{N^{\mathrm{in}}}, [-1,1]^{N^{\mathrm{hidden}}}\right) &\to [-1,1]^{N^{\mathrm{hidden}}} \\ (\underline{x}, \underline{h}) &\mapsto \overset{\odot}{\tanh}\left[\underline{\underline{A}}_x^{\mathrm{c}} \underline{x} + \underline{b}_x^{\mathrm{c}} + \mathcal{F}^{\mathrm{r}}(\underline{x}, \underline{h}) \odot \left(\underline{\underline{A}}_h^{\mathrm{c}} \underline{h} + \underline{b}_h^{\mathrm{c}}\right)\right], \end{aligned} \tag{2.24}$$

which has the trainable weight matrices and bias vectors

$$\begin{aligned} \underline{\underline{A}}_x^{\mathrm{c}} &\in \mathbb{R}^{N^{\mathrm{hidden}} \times N^{\mathrm{in}}}, & \underline{b}_x^{\mathrm{c}} &\in \mathbb{R}^{N^{\mathrm{hidden}}}, \\ \underline{\underline{A}}_h^{\mathrm{c}} &\in \mathbb{R}^{N^{\mathrm{hidden}} \times N^{\mathrm{hidden}}}, & \underline{b}_h^{\mathrm{c}} &\in \mathbb{R}^{N^{\mathrm{hidden}}}. \end{aligned} \tag{2.25}$$

The above map to the candidate state resembles the maps to the reset and the update gate by subjecting the current data point and the previous hidden state to a separate, biased linear transformation, but differs in two aspects. First, before adding the results of the two transformations, the current reset gate applies to the transformed, previous hidden state ($\odot$ denotes the Hadamard product). Thereby, the reset gate mitigates the contribution of the previous hidden state to the candidate state. Second, instead of the sigmoid function, the hyperbolic tangent [52] (see fig. 2.2)

$$\tanh : \mathbb{R} \to [-1, 1] \,;\; x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.26}$$

applies element-wise to the sum of the transformed data point and the gated, transformed, previous hidden state, which limits the values of the candidate state to the interval from minus to plus one.

Finally, the GRU layer computes the current hidden state

$$h_{t,i} = \mathcal{F}^{\mathrm{h}} \left(\underline{x}_{t,i}, \underline{h}_{t-1,i} \right), \quad i \in \left\{ 1, \ldots, N^{\mathrm{batch}} \right\} \tag{2.27}$$

with the map

$$\begin{aligned} \mathcal{F}^{\mathrm{h}} &: \left(\mathbb{R}^{N^{\mathrm{in}}},\, [-1, 1]^{N^{\mathrm{hidden}}} \right) \to [-1, 1]^{N^{\mathrm{hidden}}} \\ &\chi := (\underline{x},\, \underline{h}) \mapsto [\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)] \odot \mathcal{F}^{\mathrm{c}}(\chi) + \mathcal{F}^{\mathrm{u}}(\chi) \odot \underline{h}. \end{aligned} \tag{2.28}$$

The above map to the hidden state is a weighted arithmetic mean of the previous hidden state and the current candidate state, where the current update gate and its counter gate are the weights. Therewith, the update gate controls the element-wise percentages of the two states in the current hidden state. The fact that the hyperbolic tangent normalizes the elements of the candidate state to the interval from minus to plus one (see equ. 2.24) limits the values of the current hidden state to the same interval (given that the values of initial hidden state (equ. 2.14) are also in that interval).

Backpropagation through Time

The following illustrates backpropagation through time (BPTT) applied to a single integrated GRU layer, which operates in many-to-one mode. BPTT calculates the gradients of the loss with respect to the trainable parameters of the GRU. Learning algorithms update the trainable parameters based on these gradients to minimize the loss.

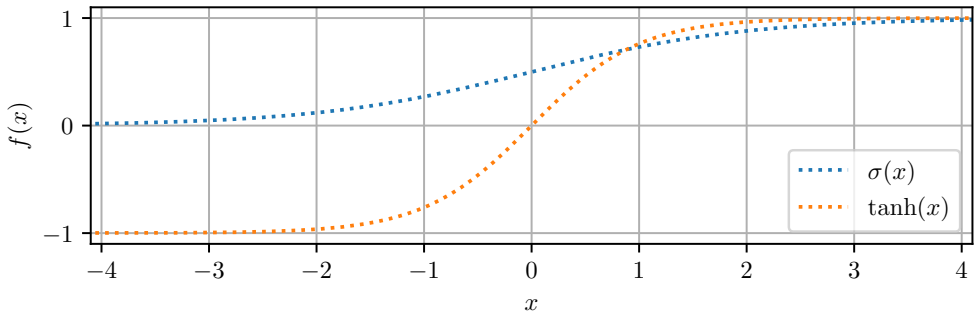


Figure 2.2: Activation functions of the GRU. The standard sigmoid function (see equ. 2.19) normalizes the values of the reset and the update gate to the interval from zero to one (see equ. 2.17 and 2.21). The hyperbolic tangent (see equ. 2.26) normalizes the values of the candidate and therewith the hidden state to the interval from minus to plus one (see equ. 2.24 and 2.28).

Let a single GRU layer, integrated into a superordinate ANN, o perate in many-to-one mode. Given biases enabled, the set of all objects of trainable parameters is

$$\Theta = \left\{ \underline{\underline{A}}_x^{\mathrm{r}}, \underline{b}_x^{\mathrm{r}}, \underline{\underline{A}}_h^{\mathrm{r}}, \underline{b}_h^{\mathrm{r}}, \underline{\underline{A}}_x^{\mathrm{u}}, \underline{b}_x^{\mathrm{u}}, \underline{\underline{A}}_h^{\mathrm{u}}, \underline{b}_h^{\mathrm{u}}, \underline{\underline{A}}_x^{\mathrm{c}}, \underline{b}_x^{\mathrm{c}}, \underline{\underline{A}}_h^{\mathrm{c}}, \underline{b}_h^{\mathrm{c}} \right\} \tag{2.29}$$

(see equ. 2.18, 2.22 and 2.25). The total number of trainable parameters of the GRU layer is

$$N^{\mathrm{param}} = \begin{cases} 3N^{\mathrm{hidden}} \left(N^{\mathrm{in}} + N^{\mathrm{hidden}} + 2 \right), & \text{if biases enabled} \\ 3N^{\mathrm{hidden}} \left(N^{\mathrm{in}} + N^{\mathrm{hidden}} \right), & \text{else.} \end{cases} \tag{2.30}$$

At a single inference, the GRU layer inputs a batch of sequences of feature vectors from the precedent layer of the ANN

$$\left(\underline{x}_t \right)_{t \in \{1,\ldots,N^{\mathrm{seq}}\},i}, \quad i \in \left\{ 1, \ldots, N^{\mathrm{batch}} \right\}. \tag{2.31}$$

The GRU layer maps each incoming batch of sequences to a batch of single outputs, whereby a single output is the hidden state from the last processing step in the sequence

$$\underline{y}_i = \underline{h}_{N^{seq},i}, \quad i \in \left\{ 1, \ldots, N^{\mathrm{batch}} \right\}. \tag{2.32}$$

Figure 2.3 shows the time-unfolded computation graph of the integrated GRU layer for a single batch element. The GRU layer forwards the batches of single outputs to the subsequent layer of the ANN. After passing the output layer of the ANN, the loss of the batch is computed by averaging the losses of the batch elements

$$L\left(\underline{y}_1, \ldots, \underline{y}_{N^{\mathrm{batch}}} \right) = \frac{1}{N^{\mathrm{batch}}} \sum_{i=1}^{N^{\mathrm{batch}}} L_i \left(\underline{y}_i \right). \tag{2.33}$$

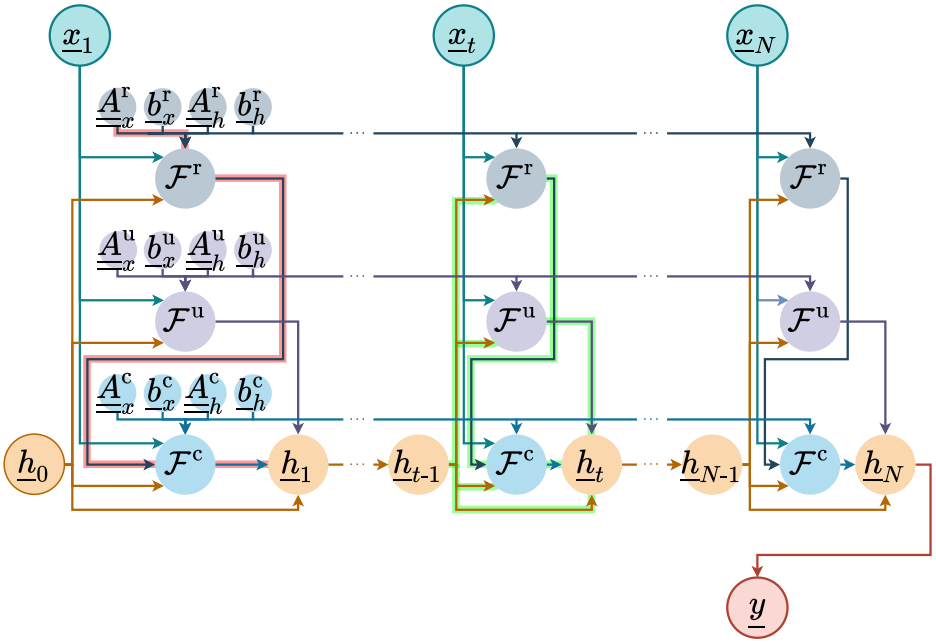


Figure 2.3: Time-unfolded computation graph of a GRU layer in many-to-one mode, which maps the input sequence $(x_t)_{t\in\{1,\dots N\}}$ to the single output $\underline{y}$ equal to the hidden state $\underline{h}_N$ of the last processing step. $\mathcal{F}^{\mathrm{r}}$, $\mathcal{F}^{\mathrm{u}}$ and $\mathcal{F}^{\mathrm{c}}$ (see equ. 2.17, 2.21 and 2.24) are the maps to the reset gate, update gate and candidate state, respectively. The map $\mathcal{F}^{\mathrm{h}}$ (see equ. 2.28) yields the hidden states $(h_t)_{t\in\{1,\dots N\}}$, whereas $\underline{h}_0$ is initialized. The trainable objects $\underline{\underline{A}}^{\square}_{\square}, \underline{b}^{\square}_{\square}$ of the GRU layer are shared across all unfolded time steps. The backpropagation path of the intra-gradient with respect to $\underline{\underline{A}}^{\mathrm{r}}_{x}$ (see equ. 2.36) is highlighted in red for $t = 1$. The backpropagation path of the inter-gradient (see equ. 2.39) is highlighted in green for t.

Gradient-based methods update the trainable parameters of the ANN with the goal to minimize the batch loss. To do this, they compute the gradients of the batch loss with respect to the trainable parameters. The update of the trainable parameters of the integrated GRU layer complies with

$$\underset{\Theta}{\operatorname{argmin}}\ L\left(\underline{y}_1, \ldots, \underline{y}_{N^{\text{batch}}}\right). \tag{2.34}$$

In conformity with Li [34], BPTT calculates the gradient of the batch loss with respect to an element $\theta \in \Theta$ in the set of trainable objects (see equ. 2.29)

$$\begin{aligned}
&\frac{\partial}{\partial\theta} L\left(\underline{y}_1, \ldots, \underline{y}_{N^{\text{batch}}}\right) \\
&\overset{(1)}{=} \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} \frac{\partial}{\partial\theta} L_i\left(\underline{y}_i\right) \\
&\overset{(2)}{=} \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} \left(\frac{\partial L_i}{\partial \underline{y}_i} \frac{\partial \underline{y}_i}{\partial\theta}\right) \\
&\overset{(3)}{=} \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} \left[\frac{\partial L_i}{\partial \underline{y}_i} \sum_{j=1}^{N^{seq}} \left(\frac{\partial \underline{y}_i}{\partial \underline{h}_{j,i}} \widehat{\frac{\partial \underline{h}_{j,i}}{\partial\theta}}\right)\right] \\
&\overset{(4)}{=} \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} \left\{\frac{\partial L_i}{\partial \underline{y}_i} \sum_{j=1}^{N^{seq}} \left[\prod_{k=j+1}^{N^{seq}} \left(\frac{\partial \underline{h}_{k,i}}{\partial \underline{h}_{k-1,i}}\right) \widehat{\frac{\partial \underline{h}_{j,i}}{\partial\theta}}\right]\right\}.
\end{aligned} \tag{2.35}$$

(1) Batch loss (equ. 2.33)

(2) Chain rule

(3) BPTT (gradient $\widehat{\square}$ considers previous hidden states constant)

(4) Many-to-one (equ. 2.32); chain rule applied to $\partial \underline{y}_i / \partial \underline{h}_{j,i}$

In the above formula, the gradient $\widehat{\partial \underline{h}_{j,i} / \partial\theta}$ (hereinafter referred to as intra-gradient) treats all previous hidden states $\underline{h}_{\tilde{j},\ldots,i},\ \tilde{j} \in \{0, j-1\}$ as constants. In doing so, the intra-gradient covers the backpropagation path from the hidden state $\underline{h}_{j,i}$ to the trainable object θ only within the same time step j. The full gradient covering all backpropagation paths to θ, connects all intra-backpropagation paths through time to the output $\underline{y}_i$ by multiplying the intra-gradients with the corresponding chains of time step inter-gradients $\partial \underline{h}_{k,i} / \partial \underline{h}_{k-1,i}$.

The following exemplarily derives the intra-gradient with respect to $\theta = \underline{\underline{A}}^{\text{r}}_x$ (see fig

2.3 for the corresponding backpropagation path):

$$\begin{aligned}\widehat{\frac{\partial \underline{h}_{t,i}}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}}} &\overset{(1)}{=} \widehat{\frac{\partial}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}}} \left\{ [\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)] \odot \mathcal{F}^{\mathrm{c}}(\chi) + \mathcal{F}^{\mathrm{u}}(\chi) \odot \underline{h}_{t\text{-}1,i} \right\} \\ &\overset{(2)}{=} \mathrm{diag}\left[\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)\right] \widehat{\frac{\partial \mathcal{F}^{\mathrm{c}}(\chi)}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}}} \end{aligned} \tag{2.36}$$

(1) Map to hidden state (equ. 2.27, 2.28); $\chi := \left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)$

(2) Sum rule;

Map to update gate (equ. 2.21): $\widehat{\partial \mathcal{F}^{\mathrm{u}} / \partial \underline{\underline{A}}^{\mathrm{r}}_{x}} = 0$

Consider $\widehat{\partial \underline{h}_{t\text{-}1,i} / \partial \underline{\underline{A}}^{\mathrm{r}}_{x}} = 0$;

Hadamard product of 2 vectors (equ. 2.45)

with

$$\begin{aligned}\widehat{\frac{\partial \mathcal{F}^{\mathrm{c}}(\chi)}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}}} &\overset{(1)}{=} \widehat{\frac{\partial}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}}} \left\{ \overset{\odot}{\tanh} \left[\underbrace{\underline{\underline{A}}^{\mathrm{c}}_{x} \underline{x}_{t,i} + \underline{b}^{\mathrm{c}}_{x} + \mathcal{F}^{\mathrm{r}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right) \odot \left(\underline{\underline{A}}^{\mathrm{c}}_{h} \underline{h}_{t\text{-}1,i} + \underline{b}^{\mathrm{c}}_{h}\right)}_{:=\chi^{\mathrm{c}}} \right] \right\} \\ &\overset{(2)}{=} \mathrm{diag}\left[\frac{\partial}{\partial \chi^{\mathrm{c}}} \overset{\odot}{\tanh}(\chi^{\mathrm{c}}) \right] \cdot \widehat{\frac{\partial \chi^{\mathrm{c}}}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}}} \\ &\overset{(3)}{=} \mathrm{diag}\left[1 - \overset{\odot}{\tanh}^{2}(\chi^{\mathrm{c}}) \right] \cdot \mathrm{diag}\left(\underline{\underline{A}}^{\mathrm{c}}_{h} \underline{h}_{t-1,i} + \underline{b}^{\mathrm{c}}_{h} \right) \widehat{\frac{\partial \mathcal{F}^{\mathrm{r}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}}} \end{aligned} \tag{2.37}$$

(1) Map to candidate state (equ. 2.24); $\chi := \left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)$

(2) Chain rule;

Derivative of element-wise function (equ. 2.47)

(3) Derivative of tanh (equ. 2.43);

Hadamard product of 2 vectors (equ. 2.45)

and

$$\begin{aligned}\frac{\partial \mathcal{F}^{\mathrm{r}}\left(\widehat{\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}}\right)}{\partial \underline{\underline{A}}_{x}^{\mathrm{r}}} &\overset{(1)}{=} \widehat{\frac{\partial}{\partial \underline{\underline{A}}_{x}^{\mathrm{r}}}} \left[\overset{\odot}{\sigma}\left(\underbrace{\underline{\underline{A}}_{x}^{\mathrm{r}} \underline{x}_{t,i} + \underline{b}_{x}^{\mathrm{r}} + \underline{\underline{A}}_{h}^{\mathrm{r}} \underline{h}_{t\text{-}1,i} + \underline{b}_{h}^{\mathrm{r}}}_{:=\chi^{\mathrm{r}}}\right)\right] \\ &\overset{(2)}{=} \operatorname{diag}\left[\frac{\partial}{\partial \chi^{\mathrm{r}}} \overset{\odot}{\sigma}\left(\chi^{\mathrm{r}}\right)\right] \cdot \widehat{\frac{\partial \chi^{\mathrm{u}}}{\partial \underline{\underline{A}}_{x}^{\mathrm{r}}}} \\ &\overset{(3)}{=} \operatorname{diag}\left\{\overset{\odot}{\sigma}\left(\chi^{\mathrm{r}}\right) \odot \left[1 - \overset{\odot}{\sigma}\left(\chi^{\mathrm{r}}\right)\right]\right\} \cdot \frac{\partial \underline{\underline{A}}_{x}^{\mathrm{r}} \underline{x}_{t,i}}{\partial \underline{\underline{A}}_{x}^{\mathrm{r}}}. \end{aligned} \tag{2.38}$$

(1) Map to reset gate (equ. 2.17)

(2) Chain rule;

Derivative of element-wise function (equ. 2.47)

(3) Derivative of sigmoid (equ. 2.44)

For the calculation of the gradient $\partial\left(\underline{\underline{A}}_{x}^{\mathrm{r}} \underline{x}_{t,i}\right) / \partial \underline{\underline{A}}_{x}^{\mathrm{r}}$, which is a three-dimensional tensor of only two-dimensional information content, refer to [33], for example.

As required by equation 2.35, the following derives the inter-gradient (see fig 2.3 for the corresponding backpropagation path):

$$\begin{aligned}\frac{\partial \underline{h}_{t,i}}{\partial \underline{h}_{t\text{-}1,i}} &\overset{(1)}{=} \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \left\{[\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)] \odot \mathcal{F}^{\mathrm{c}}(\chi) + \mathcal{F}^{\mathrm{u}}(\chi) \odot \underline{h}_{t\text{-}1,i}\right\} \\ &\overset{(2)}{=} \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \left\{[\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)] \odot \mathcal{F}^{\mathrm{c}}(\chi)\right\} + \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \left\{\mathcal{F}^{\mathrm{u}}(\chi) \odot \underline{h}_{t-1,i}\right\} \\ &\overset{(3)}{=} -\operatorname{diag}\left[\mathcal{F}^{\mathrm{c}}(\chi)\right] \frac{\partial \mathcal{F}^{\mathrm{u}}(\chi)}{\partial \underline{h}_{t\text{-}1,i}} + \operatorname{diag}\left[\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)\right] \frac{\partial \mathcal{F}^{\mathrm{c}}(\chi)}{\partial \underline{h}_{t\text{-}1,i}} \\ &\quad + \operatorname{diag}\left(\underline{h}_{t-1,i}\right) \frac{\partial \mathcal{F}^{\mathrm{u}}(\chi)}{\partial \underline{h}_{t\text{-}1,i}} + \operatorname{diag}\left[\mathcal{F}^{\mathrm{u}}(\chi)\right]. \end{aligned} \tag{2.39}$$

(1) Map to hidden state (equ. 2.27, 2.28); $\chi := \left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)$

(2) Sum rule

(3) Differentiation of Hadamard product of 2 vectors (equ. 2.46)

with

$$\begin{aligned}\frac{\partial \mathcal{F}^{\mathrm{u}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)}{\partial \underline{h}_{t\text{-}1,i}} &\overset{(1)}{=} \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}}\left[\overset{\odot}{\sigma}\left(\underbrace{\underline{\underline{A}}_x^{\mathrm{u}}\underline{x}_{t,i} + \underline{b}_x^{\mathrm{u}} + \underline{\underline{A}}_h^{\mathrm{u}}\underline{h}_{t\text{-}1,i} + \underline{b}_h^{\mathrm{u}}}_{:=\chi^{\mathrm{u}}}\right)\right] \\ &\overset{(2)}{=} \operatorname{diag}\left[\frac{\partial}{\partial \chi^{\mathrm{u}}}\overset{\odot}{\sigma}\left(\chi^{\mathrm{u}}\right)\right] \cdot \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}}\chi^{\mathrm{u}} \\ &\overset{(3)}{=} \operatorname{diag}\left\{\overset{\odot}{\sigma}\left(\chi^{\mathrm{u}}\right) \odot \left[1 - \overset{\odot}{\sigma}\left(\chi^{\mathrm{u}}\right)\right]\right\} \cdot \underline{\underline{A}}_h^{\mathrm{u}}, \end{aligned} \tag{2.40}$$

(1) Map to update gate (equ. 2.21)

(2) Chain rule;

Derivative of element-wise function (equ. 2.47)

(3) Derivative of sigmoid (equ. 2.44)

and

$$\begin{aligned}&\frac{\partial}{\partial \underline{h}_{t\text{-}1,i}}\mathcal{F}^{\mathrm{c}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right) \\ &\quad\overset{(1)}{=} \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}}\left\{\overset{\odot}{\tanh}\left[\underbrace{\underline{\underline{A}}_x^{\mathrm{c}}\underline{x}_{t,i} + \underline{b}_x^{\mathrm{c}} + \mathcal{F}^{\mathrm{r}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right) \odot \left(\underline{\underline{A}}_h^{\mathrm{c}}\underline{h}_{t\text{-}1,i} + \underline{b}_h^{\mathrm{c}}\right)}_{:=\chi^{\mathrm{c}}}\right]\right\} \\ &\quad\overset{(2)}{=} \operatorname{diag}\left[\frac{\partial}{\partial \chi^{\mathrm{c}}}\overset{\odot}{\tanh}\left(\chi^{\mathrm{c}}\right)\right] \cdot \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}}\chi^{\mathrm{c}} \\ &\quad\overset{(3)}{=} \operatorname{diag}\left[1 - \overset{\odot}{\tanh}^2\left(\chi^{\mathrm{c}}\right)\right] \\ &\qquad \cdot \left\{\operatorname{diag}\left(\underline{\underline{A}}_h^{\mathrm{c}}\underline{h}_{t-1,i} + \underline{b}_h^{\mathrm{c}}\right)\frac{\partial \mathcal{F}^{\mathrm{r}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)}{\partial \underline{h}_{t\text{-}1,i}} + \operatorname{diag}\left[\mathcal{F}^{\mathrm{r}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)\right]\underline{\underline{A}}_h^{\mathrm{c}}\right\}. \end{aligned} \tag{2.41}$$

(1) Map to candidate state (equ. 2.24)

(2) Chain rule of differentiation;

Derivative of function applied element-wise on vector (equ. 2.47)

(3) Derivative of hyperbolic tangent (equ. 2.43);

Differentiation of Hadamard product of 2 vectors (equ. 2.46)

as well as (same structure as in equ. 2.40)

$$\frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \mathcal{F}^{\mathrm{r}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right) = \operatorname{diag}\left\{\overset{\odot}{\sigma}\left(\chi^{\mathrm{r}}\right) \odot \left[1 - \overset{\odot}{\sigma}\left(\chi^{\mathrm{r}}\right)\right]\right\} \cdot \underline{\underline{A}}_{h}^{\mathrm{r}}$$
$$\text{with } \chi^{\mathrm{r}} := \underline{\underline{A}}_{x}^{\mathrm{r}} \underline{x}_{t,i} + \underline{b}_{x}^{\mathrm{r}} + \underline{\underline{A}}_{h}^{\mathrm{r}} \underline{h}_{t\text{-}1,i} + \underline{b}_{h}^{\mathrm{r}}. \tag{2.42}$$

The above derivations revert to the following equations. Equations (4.5.73) and (4.5.17) of Abramowitz and Stegun [52] yield the derivative of the hyperbolic tangent as

$$\frac{\mathrm{d}}{\mathrm{d}x} \tanh(x) = 1 - \tanh^2(x). \tag{2.43}$$

The derivative of the sigmoid function (e.g., [41]) is

$$\frac{\mathrm{d}}{\mathrm{d}x} \sigma\left(x\right) = \sigma\left(x\right)\left[1 - \sigma\left(x\right)\right]. \tag{2.44}$$

The Hadamard product of two vectors [6] equates to the matrix product of a diagonal matrix and a vector

$$\underline{x}_1 \odot \underline{x}_2 = \operatorname{diag}\left(\underline{x}_1\right) \underline{x}_2 = \operatorname{diag}\left(\underline{x}_2\right) \underline{x}_1. \tag{2.45}$$

Applying the product rule of differentiation yields the derivative of the Hadamard product of two vectors

$$\frac{\partial}{\partial t}\left(\underline{x}_1 \odot \underline{x}_2\right) = \operatorname{diag}\left(\underline{x}_2\right) \frac{\partial \underline{x}_1}{\partial t} + \operatorname{diag}\left(\underline{x}_1\right) \frac{\partial \underline{x}_2}{\partial t}. \tag{2.46}$$

The derivative of a function $f : \mathbb{R} \rightarrow \mathbb{R}$ applied element-wise to a vector $\underline{x}$ is the diagonal matrix built from the derivative of the function applied element-wise on the vector

$$\frac{\partial}{\partial \underline{x}} \overset{\odot}{f}\left(\underline{x}\right) = \operatorname{diag}\left[\overset{\odot}{f}'\left(\underline{x}\right)\right]. \tag{2.47}$$

The calculation of the differential of the function yields the above statement

$$\frac{\mathrm{d}}{\mathrm{d}\underline{x}} \overset{\odot}{f}\left(\underline{x}\right) \cdot \mathrm{d}\underline{x} \overset{(1)}{=} \mathrm{d}\overset{\odot}{f}\left(\underline{x}\right) \overset{(2)}{=} \left[\overset{\odot}{f}'\left(\underline{x}\right)\right] \odot \mathrm{d}\underline{x} \overset{(3)}{=} \operatorname{diag}\left[\overset{\odot}{f}'\left(\underline{x}\right)\right] \cdot \mathrm{d}\underline{x}. \tag{2.48}$$

(1) Relation of differential and derivative

(2) Chain rule element-wise applied

(3) Hadamard product of 2 vectors (equ. 2.45) (2.49)

Chapter 3

Autonomous Navigation Method

This chapter presents the autonomous drone racing method of Kaufmann et al. [31], which is used as a baseline for the experiments of this thesis. The first section introduces the three reference systems and their transformations applied by the modules of the method. The second section presents the ANN module of this thesis, which has the function of making navigation decisions based on the RGB images from the drone's onboard camera. The ANN module, comprising the CNN, CAT, FC, GRU and HEAD submodules, is a modularized version of the ANN of the baseline work. It additionally integrates the CAT submodule, which extends the decision-making basis with the optional inputs (i.e., the time steps of the images and the estimates from the drone's onboard IMU), and the GRU submodule, which extends the decision-making capabilities with temporal comprehension. The third section presents the planning module of the method, which has the function of computing local trajectories based on the navigation decisions made by the ANN module or, possibly in the imitation learning process, the expert system. The fourth section presents the control stack of the method, which has the function of computing the drone's motor inputs to track the local trajectories computed by the planning module. The fifth section presents the expert system of the method, which in the rollouts of the imitation learning process, intervenes and generates a training sample whenever the ANN module makes a poor navigation decision.

3.1 Reference Systems

In this thesis, the coordinates of a point $\underline{p}$ relate to either the global, the local or the image reference system

$$ {}_{\mathbf{G}}\underline{p} = \begin{bmatrix} {}_{\mathbf{G}}x \\ {}_{\mathbf{G}}y \\ {}_{\mathbf{G}}z \end{bmatrix} \in \mathbb{R}^3, \quad {}_{\mathbf{L}}\underline{p} = \begin{bmatrix} {}_{\mathbf{L}}x \\ {}_{\mathbf{L}}y \\ {}_{\mathbf{L}}z \end{bmatrix} \in \mathbb{R}^3, \quad {}_{\mathbf{I}}\underline{p} = \begin{bmatrix} {}_{\mathbf{I}}x \\ {}_{\mathbf{I}}y \end{bmatrix} \in [\text{-}1, 1]^2 . \tag{3.1} $$

The 3D global reference system is fixed to an arbitrary point on earth and is hence quasi inertial. It is spanned by the orthonormal basis, which, related to the global reference system, equates to the standard basis of $\mathbb{R}^3$

$$\left\{\underline{e}_x^{\mathrm{G}}, \underline{e}_y^{\mathrm{G}}, \underline{e}_z^{\mathrm{G}}\right\} \quad \text{with} \quad {}_{\mathbf{G}}\underline{e}_x^{\mathrm{G}} = \begin{bmatrix}1\\0\\0\end{bmatrix}, \; {}_{\mathbf{G}}\underline{e}_y^{\mathrm{G}} = \begin{bmatrix}0\\1\\0\end{bmatrix}, \; {}_{\mathbf{G}}\underline{e}_z^{\mathrm{G}} = \begin{bmatrix}0\\0\\1\end{bmatrix}. \tag{3.2}$$

The local reference system (see fig. 3.1a) is fixed on the moving drone. It is spanned by the orthonormal basis, whose origin is located at the optical center of the drone's onboard camera

$$\left\{\underline{e}_x^{\mathrm{L}}, \underline{e}_y^{\mathrm{L}}, \underline{e}_z^{\mathrm{L}}\right\} \quad \text{with} \quad {}_{\mathbf{L}}\underline{e}_x^{\mathrm{L}} = \begin{bmatrix}1\\0\\0\end{bmatrix}, \; {}_{\mathbf{L}}\underline{e}_y^{\mathrm{L}} = \begin{bmatrix}0\\1\\0\end{bmatrix}, \; {}_{\mathbf{L}}\underline{e}_z^{\mathrm{L}} = \begin{bmatrix}0\\0\\1\end{bmatrix}. \tag{3.3}$$

The unit vector $\underline{e}_x^{\mathrm{L}}$ points along the optical axis of the camera in the flight direction of the drone. The unit vector $\underline{e}_z^{\mathrm{L}}$ points in the direction of the forces generated by the drone's rotors and is parallel to the vertical axis of the image plane of the drone's onboard camera. The unit vector $\underline{e}_y^{\mathrm{L}}$ points to the left of the drone and parallels the horizontal axis of the image plane.

The image reference system (see fig. 3.1b) is superimposed on the images of the drone's onboard camera. This 2-dimensional system is spanned by the orthonormal basis

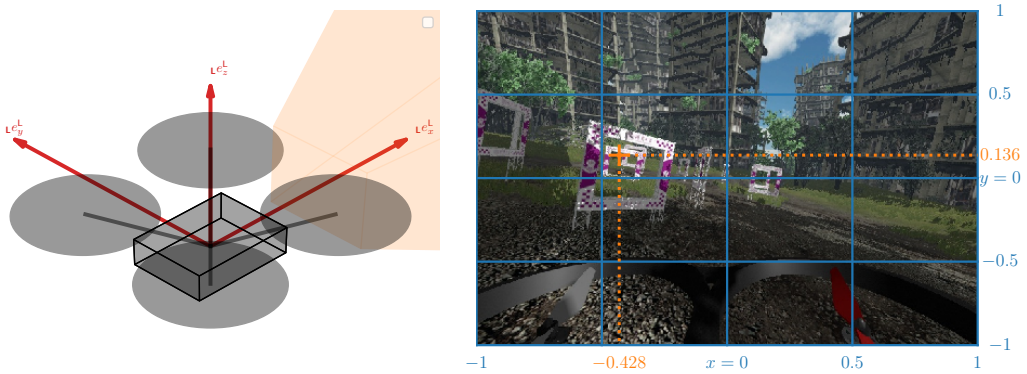
$$\left\{\underline{e}_x^{\mathrm{I}}, \underline{e}_y^{\mathrm{I}}\right\} \quad \text{with} \quad {}_{\mathbf{I}}\underline{e}_x^{\mathrm{I}} = \begin{bmatrix}1\\0\end{bmatrix}, \; {}_{\mathbf{I}}\underline{e}_y^{\mathrm{I}} = \begin{bmatrix}0\\1\end{bmatrix}. \tag{3.4}$$

The origin of the image reference system is located at the center of the image plane. The unit vector $\underline{e}_x^{\mathrm{I}}$ points rightwards along the vertical axis of the image plane. The unit vector $\underline{e}_y^{\mathrm{I}}$ points upwards along the horizontal axis of the image plane. A point on the image plane is bounded by the left and right $-1 \leq {}_{\mathbf{I}}x \leq 1$ as well as the lower and upper $-1 \leq {}_{\mathbf{I}}y \leq 1$ border of the image plane.

Global-local Transformations

The drone's position ${}_{\mathbf{G}}\underline{p}^{\mathrm{d}}$ and quaternion orientation ${}_{\mathbf{G}}\underline{q}^{\mathrm{d}}$ with respect to the global reference system are the parameters that determine the bidirectional transformation between the global and the local reference system. The following bases on quaternion mathematics, for which one can consult, e.g., [46]. A point given in the coordinates of the global reference system can be expressed in the coordinates of the local reference system with the transformation

$$T_{\mathbf{LG}} : \mathbb{R}^3 \to \mathbb{R}^3; \quad {}_{\mathbf{G}}\underline{p} \mapsto {}_{\mathbf{L}}\underline{p} = \mathcal{P}\left[\mathrm{inv}\left({}_{\mathbf{G}}\underline{q}^{\mathrm{d}}\right) * \mathcal{Q}\left({}_{\mathbf{G}}\underline{p} - {}_{\mathbf{G}}\underline{p}^{\mathrm{d}}\right) * {}_{\mathbf{G}}\underline{q}^{\mathrm{d}}\right], \tag{3.5}$$



(a) Local reference system (b) Image reference system

Figure 3.1: The local reference system (red), which is aligned with the drone's onboard camera, and the image reference system (blue), which is superimposed on the images from the onboard camera. The pictured, exemplary waypoint ${}_{\mathbf{I}}\underline{p}^{\mathrm{wp}} = \begin{bmatrix} -0.428 & 0.136 \end{bmatrix}^T$ (orange) is part of the expert navigation decision labeling the underlying image.

Reversely, a point given in the coordinates of the local reference system can be expressed in the coordinates of the global reference system with the transformation

$$T_{\mathbf{GL}} : \mathbb{R}^3 \to \mathbb{R}^3; \quad {}_{\mathbf{L}}\underline{p} \mapsto {}_{\mathbf{G}}\underline{p} = \mathcal{P}\left[{}_{\mathbf{G}}\underline{q}^{\mathrm{d}} * \mathcal{Q}\left({}_{\mathbf{L}}\underline{p}\right) * \mathrm{inv}\left({}_{\mathbf{G}}\underline{q}^{\mathrm{d}}\right)\right] + {}_{\mathbf{G}}\underline{p}^{\mathrm{d}}. \tag{3.6}$$

In the two above transformations, the mapping $\mathcal{Q}$ of a point to its quaternion representation and the reverse mapping $\mathcal{P}$ of a quaternion representation to its point are given by

$$\begin{aligned} \mathcal{Q} : \mathbb{R}^3 \to \mathbb{R}^4; \ \underline{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \mapsto \underline{q} = \begin{bmatrix} w \\ \underline{p} \end{bmatrix} \ \text{with } w = 0 \\ \mathcal{P} : \mathbb{R}^4 \to \mathbb{R}^3; \ \underline{q} = \begin{bmatrix} w \\ \underline{p} \end{bmatrix} \mapsto \underline{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \end{aligned} \tag{3.7}$$

Moreover, the operator $*$ denotes the multiplication of two quaternions, which is given by

$$\underline{q}_1 * \underline{q}_2 = \begin{bmatrix} w_1 w_2 - \underline{p}_1^T \underline{p}_2 \\ w_1 \underline{p}_2 + w_2 \underline{p}_1 + \underline{p}_1 \times \underline{p}_2 \end{bmatrix}. \tag{3.8}$$

Finally, the inversion of a quaternion is given by

$$\mathrm{inv}(\underline{q}) = \frac{1}{\|\underline{q}\|_2} \begin{bmatrix} w \\ \text{-}\underline{p} \end{bmatrix}. \tag{3.9}$$

The two above transformations are the inversion of each other. Therefore, points can be transformed between the global and local reference system without information loss

$$T_{\mathbf{GL}} \circ T_{\mathbf{LG}} \left({}_{\mathbf{G}}\underline{p} \right) = {}_{\mathbf{G}}\underline{p}, \quad T_{\mathbf{LG}} \circ T_{\mathbf{GL}} \left({}_{\mathbf{L}}\underline{p} \right) = {}_{\mathbf{L}}\underline{p}. \tag{3.10}$$

In the above equations, the operator $\circ$ denotes the composition of two functions.

Local-image Transformations

The horizontal $\breve{\phi}_{\mathrm{h}}^{\mathrm{cam}}$ and the vertical $\breve{\phi}_{\mathrm{v}}^{\mathrm{cam}}$ angle of view of the drone's onboard camera are the parameters that determine the bidirectional transformation between the local and the image reference system. A point given in the coordinates of the local reference system is expressed in the coordinates of the image reference system with the transformation

$$T_{\mathbf{IL}} : \mathbb{R}^3 \to [\text{-}1, 1]^2; \quad {}_{\mathbf{L}}\underline{p} \mapsto {}_{\mathbf{I}}\underline{p} = \begin{bmatrix} \max\left(\text{-}1,\ \min\left(\frac{\text{-}2}{\breve{\phi}_{\mathrm{h}}^{\mathrm{cam}}} \mathrm{atan2}\left({}_{\mathbf{L}}y, {}_{\mathbf{L}}x\right),\ 1\right)\right) \\ \max\left(\text{-}1,\ \min\left(\frac{2}{\breve{\phi}_{\mathrm{v}}^{\mathrm{cam}}} \mathrm{atan2}\left({}_{\mathbf{L}}z, \|{}_{\mathbf{L}}\underline{p}\|_2\right),\ 1\right)\right) \end{bmatrix}. \tag{3.11}$$

The above transformation can be interpreted as the projection of a point onto the image plane of the drone's onboard camera. It can be divided into three steps. First, the vector from the optical center of the camera to the point to be transformed is mapped to its yaw $\mathrm{atan2}\left({}_{\mathbf{L}}y, {}_{\mathbf{L}}x\right)$ and pitch $\mathrm{atan2}\left({}_{\mathbf{L}}z, \|{}_{\mathbf{L}}\underline{p}\|_2\right)$ angle, both, with respect to the image reference system. Second these angles are normalized by the half of the horizontal $\breve{\phi}_{\mathrm{h}}^{\mathrm{cam}}$ and the half of the vertical $\breve{\phi}_{\mathrm{v}}^{\mathrm{cam}}$ angle of view of the camera, respectively. Third, these normalized angles are bounded to be in the interval from minus to plus one. This boundary takes into account that an artificial neural network, which inputs images, has no basis for predictions that relate to objects that are not within the camera's field of view. As a projection from 3D to 2D, the above transformation is accompanied by information loss and is hence not bijective.

A point given in the coordinates of the image reference system is expressed in the coordinates of the local reference system with the reverse transformation

$$T_{\mathbf{LI}} : \mathbb{R}_{\geq 0}, [\text{-}1, 1]^2 \to \mathbb{R}^3; \quad d, {}_{\mathbf{I}}\underline{p} \mapsto {}_{\mathbf{L}}\underline{p} = d \begin{bmatrix} \cos\left({}_{\mathbf{L}}\phi_y\right) \\ \cos\left({}_{\mathbf{L}}\phi_y\right) \\ \sin\left({}_{\mathbf{L}}\phi_y\right) \end{bmatrix} \odot \begin{bmatrix} \cos\left({}_{\mathbf{L}}\phi_z\right) \\ \sin\left({}_{\mathbf{L}}\phi_z\right) \\ 1 \end{bmatrix}$$

$$\text{with} \quad {}_{\mathbf{L}}\phi_z = \text{-}\frac{\breve{\phi}_{\mathrm{h}}^{\mathrm{cam}}}{2} \cdot {}_{\mathbf{I}}x, \quad {}_{\mathbf{L}}\phi_y = \frac{\breve{\phi}_{\mathrm{v}}^{\mathrm{cam}}}{2} \cdot {}_{\mathbf{I}}y. \tag{3.12}$$

In the above transformation, the operator $\odot$ denotes the Hadamard product, i.e., the element-wise product of two equally dimensioned matrices. Because the 2D coordinates

of the image reference system can only contain information about the direction of a point, the above transformation to 3D requires the additional input of a back-projection length d.

In contrast to the transformations $T_{\mathbf{LG}}$ and $T_{\mathbf{GL}}$ between the global and the local reference system, the transformations $T_{\mathbf{IL}}$ and $T_{\mathbf{LI}}$ between the local and the image reference system are not invertible. However, for relevant points located within the camera's field of view and a well chosen back-projection length, it is assumed that the transformations approximately invert each other

$$T_{\mathbf{LI}}\left[d, T_{\mathbf{IL}}\left({}_{\mathbf{L}}\underline{p}\right)\right] \approx {}_{\mathbf{L}}\underline{p}, \quad T_{\mathbf{IL}} \circ T_{\mathbf{LI}}\left(d, {}_{\mathbf{I}}\underline{p}\right) \approx {}_{\mathbf{I}}\underline{p}. \tag{3.13}$$

Global-image Transformations
The bidirectional transformations of points between the global and the image reference frame are the compositions of the transformations via the intermittent local reference system

$$\begin{aligned} T_{\mathbf{IG}} &= T_{\mathbf{IL}} \circ T_{\mathbf{LG}} : \mathbb{R}^3 \to [\text{-}1, 1]^2 \\ T_{\mathbf{GI}} &= T_{\mathbf{GL}} \circ T_{\mathbf{LI}} : \mathbb{R}_{\geq 0}, [\text{-}1, 1]^2 \to \mathbb{R}^3. \end{aligned} \tag{3.14}$$

Due to the fact that $T_{\mathbf{LG}}$ and $T_{\mathbf{GL}}$ are the inverse of each other and the assumption that $T_{\mathbf{IL}}$ and $T_{\mathbf{LI}}$ approximately invert each other within a relevant range, the above compositions are expected to also approximately invert each other within this relevant range

$$T_{\mathbf{GI}}\left[d, T_{\mathbf{IG}}\left({}_{\mathbf{G}}\underline{p}\right)\right] \approx {}_{\mathbf{G}}\underline{p}, \quad T_{\mathbf{IG}} \circ T_{\mathbf{GI}}\left(d, {}_{\mathbf{I}}\underline{p}\right) \approx {}_{\mathbf{I}}\underline{p}. \tag{3.15}$$

3.2 ANN Module

The ANN module performs the function of making navigation decisions within the autonomous navigation method. Figure 3.2 shows the information flow of this decision-making process. The ANN module infers its decisions exclusively on the basis of preprocessed data from sensors onboard the drone. It comprises the five submodules named CNN, CAT, GRU, FC and HEAD. This modular design enables a high flexibility for the experiments with the different ANN module variants in chapter 4. All variants have in common that, first, the order of the submodules is fixed and, second, the outer submodules (CNN and HEAD) are always activated. The latter ensures the minimum functionality of the ANN module of extracting visual features of the preprocessed RGB images from the drone's onboard camera and mapping them to navigation decisions. The variants can differ in that the user specifies the design parameters of the (inner and outer) individual submodules. This can include the deactivation (reduction to the identity map) of the individual inner submodules (CAT, GRU and FC). The inner submodules perform

the functions of inputting optional features (CAT), extracting temporal features (GRU) and increasing the general complexity of the ANN module (FC).

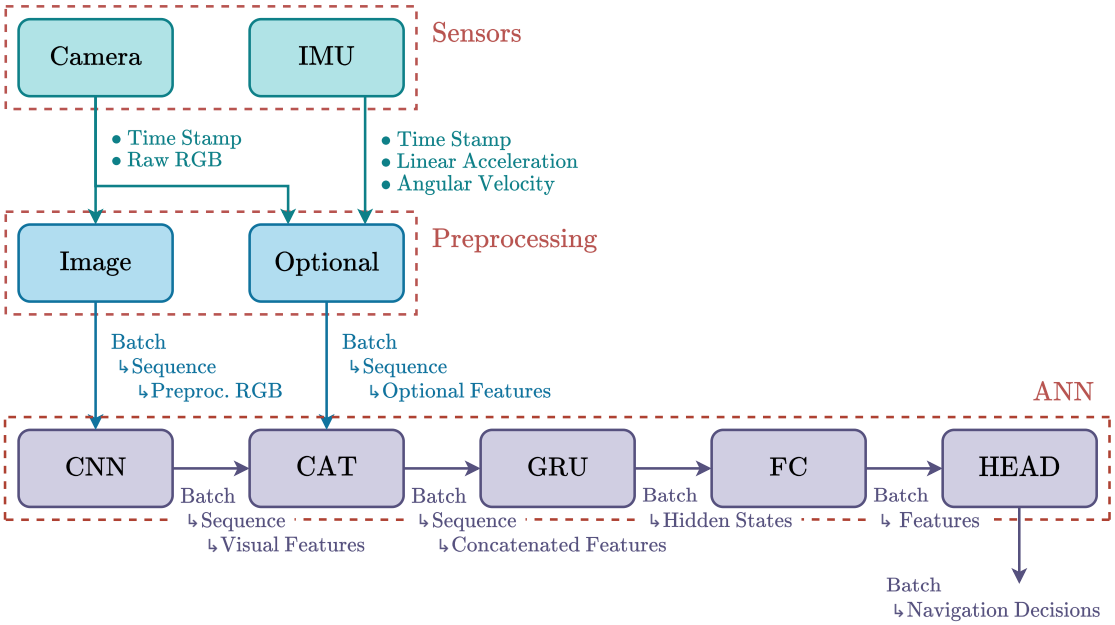


Figure 3.2: Information flow of the navigation decision-making

The ANN module learns by imitation with dataset aggregation (see section 2.1 for background). Thereby, the dataset is aggregated from samples demonstrated by the expert system (see section 3.5). Each sample is a pair of an input sequence and a single label. For the individual ANN variant, the user specifies the sequence length of the samples

$$\breve{N}^{\text{seq}} \in \mathbb{N}_{>0}. \tag{3.16}$$

The ANN module is trained on batches of the aggregated dataset with supervised learning. The user specifies the batch size

$$\breve{N}^{\text{batch}} \in \mathbb{N}_{>0}. \tag{3.17}$$

The submodules prior to the GRU submodule (CNN and CAT) have no sequential awareness and, thus, process sequence elements in parallel like batch elements. In contrast, the GRU submodule operates in many-to-one mode, whereby each input sequence of feature vectors is mapped to a single non-sequential output feature vector. The subsequent submodules (FC and HEAD) then process batches of non-sequential data, as is common in conventional feedforward networks. In this sense, the GRU submodule is mandatory in case the user specifies $\breve{N}^{\text{seq}} > 1$ for the samples of the training dataset.

At race, when the autonomous navigation method flies the drone through the race-track, the ANN module makes navigation decisions in real-time at the user-specified frequency $\breve{f}^{\text{main}}$ with the batch size and sequence length of the input $\breve{N}^{\text{batch}} = \breve{N}^{\text{seq}} = 1$. The GRU submodule (if activated in the specific variant) hence operates in one-to-one mode, i.e., each single non-sequential input feature vector is mapped to a single non-sequential output feature vector. Since the single input feature vectors trace back to the sensor data coming in at a fixed frequency, they, as a whole, constitute a time series, on which the GRU submodule can build up memory as learned during the training on the sequences of the specified length. As a result, both, the current and past sensor data, influence the current navigation decision.

Input Preprocessing

The drone's onboard camera outputs raw RGB images

$$\underline{\underline{x}}^{\text{cam}} \in \{0, \ldots, N_{\text{max}}^{\text{cam}}\}^{3 \times N_{\text{height}}^{\text{cam}} \times N_{\text{width}}^{\text{cam}}} \tag{3.18}$$

where $N_{\text{max}}^{\text{cam}}$ (usually $= 255$) is the full pixel intensity and $N_{\text{height}}^{\text{cam}} \times N_{\text{width}}^{\text{cam}}$ is the image size. The latest raw RGB image is preprocessed in two steps before being fed to the CNN submodule. First, the pixel intensities are normalized by the full intensity $N_{\text{max}}^{\text{cam}}$ with the aim to accelerate the convergence of the loss during training. Second, the image is sized down preserving its aspect ratio with the user-specified factor $\breve{s}_{\text{resize}}^{\text{cnn}}$. Besides significantly accelerating the training, this step makes training on longer sequences possible in the first place by reducing GPU memory usage. The resulting preprocessed RGB image is

$$\underline{\underline{x}}^{\text{preproc}} \in \left\{ \frac{i}{N_{\text{max}}^{\text{cam}}} \right\}_{i \in \{0, \ldots, N_{\text{max}}^{\text{cam}}\}}^{3 \times \left\lfloor \breve{s}_{\text{resize}}^{\text{cnn}} \cdot N_{\text{height}}^{\text{cam}} \right\rfloor \times \left\lfloor \breve{s}_{\text{resize}}^{\text{cnn}} \cdot N_{\text{width}}^{\text{cam}} \right\rfloor} \tag{3.19}$$

where $\lfloor \square \rfloor$ denotes the operation of rounding down to the nearest integer.

The available optional input features are

- the time step t_{Δ}^{cam} between the stamps of the raw RGB images processed at the previous and the current inference
- the latest estimate from the drone's onboard IMU (inertial measurement unit) comprising the drone's linear acceleration ${}_{\mathbf{L}}\hat{\underline{a}}^{\text{imu}} \in \mathbb{R}^3$ and angular velocity ${}_{\mathbf{L}}\hat{\underline{\omega}}^{\text{imu}} \in \mathbb{R}^3$ in the local reference system
- the time step t_{Δ}^{imu} between the stamps of the IMU estimates processed at the previous and the current inference.

The user-activated optional inputs are stacked into the optional input vector before being fed to the CAT submodule. For example, the fully activated optional input vector is

$$\underline{x}^{\text{opt}} = \begin{bmatrix} t_{\Delta}^{\text{cam}} \\ \mathbf{L}\underline{\hat{a}}^{\text{imu}} \\ \mathbf{L}\underline{\hat{\omega}}^{\text{imu}} \\ t_{\Delta}^{\text{imu}} \end{bmatrix}. \tag{3.20}$$

CNN Submodule

The CNN (convolutional neural network) submodule extracts visual features of images. This submodule is implemented with the backbone of any user-selected TorchVision classification model[1], which is obtained by removing the last layer of the model. In addition, the user specifies whether the backbone initializes with pretrained weights and whether the weights of the backbone are trainable. Since CNN backbones are only applied in this thesis, their corresponding mapping is regarded as a black box

$$\breve{\mathcal{F}}^{\text{cnn}} : \mathbb{R}^{N_{\text{channel}}^{\text{cnn}} \times N_{\text{height}}^{\text{cnn}} \times N_{\text{width}}^{\text{cnn}}} \to \mathbb{R}^{N_{\text{out}}^{\text{cnn}}}; \quad \underline{\underline{x}} \mapsto \underline{x}^{\text{vis}}. \tag{3.21}$$

The backbone implementation adapts to the inputted image in terms of the number of channels $N_{\text{channel}}^{\text{cnn}}$, height $N_{\text{height}}^{\text{cnn}}$ and width $N_{\text{width}}^{\text{cnn}}$. The backbone's output dimensionality $N_{\text{out}}^{\text{cnn}}$ is fixed by the design of its last layer. The user specifies

The CNN submodule inputs a batch of sequences of preprocessed RGB images (equ. 3.19)

$$\left(\underline{\underline{x}}_t^{\text{preproc}}\right)_{t \in \left\{1,\ldots,\breve{N}^{\text{seq}}\right\}, i}, \quad i \in \left\{1, \ldots, \breve{N}^{\text{batch}}\right\}. \tag{3.22}$$

As it processes the individual sequence elements unrelated in parallel like batch elements, the CNN submodule maps each image of the input batch to an individual visual feature vector in the output batch

$$\left(\underline{x}_t^{\text{vis}}\right)_{t \in \left\{1,\ldots,\breve{N}^{\text{seq}}\right\}, i}, \quad i \in \left\{1, \ldots, \breve{N}^{\text{batch}}\right\}. \tag{3.23}$$

The number of trainable parameters $N_{\text{params}}^{\text{cnn}}$ of the CNN submodule depends on the user-selected TorchVision model.

CAT Submodule

The CAT submodule concatenates two feature vectors

$$\mathcal{F}^{\text{cat}} : \left(\mathbb{R}^{N_1^{\text{cat}}}, \mathbb{R}^{N_2^{\text{cat}}}\right) \to \mathbb{R}^{N_1^{\text{cat}} + N_2^{\text{cat}}}; \quad (\underline{x}_1, \underline{x}_2) \mapsto \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} \tag{3.24}$$

[1] `https://pytorch.org/vision/stable/models.html`, accessed on October 5, 2022

where the input dimensionalities $\mathbb{R}^{N_1^{\text{cat}}}$ and $\mathbb{R}^{N_2^{\text{cat}}}$ adapt to the number of features of the input. This submodule applies to each visual feature vector outputted by the CNN submodule (see equ. 3.23) and optional input feature vector

$$\left(\underline{x}_t^{\text{opt}}\right)_{t\in\left\{1,\ldots,\breve{N}^{\text{seq}}\right\},i}, \quad i \in \left\{1,\ldots,\breve{N}^{\text{batch}}\right\}. \tag{3.25}$$

that correspond to the same position in batch and sequence. Hence, the CAT submodule outputs a batch of sequences of concatenated feature vectors

$$\left(\begin{bmatrix}\underline{x}_t^{\text{vis}}\\ \underline{x}_t^{\text{opt}}\end{bmatrix}\right)_{t\in\left\{1,\ldots,\breve{N}^{\text{seq}}\right\},i}, \quad i \in \left\{1,\ldots,\breve{N}^{\text{batch}}\right\}. \tag{3.26}$$

If the user deactivates all optional inputs, the CAT submodule also deactivates and recedes to the identity map of the CNN submodule output. Either way, the CAT submodule has zero trainable parameters

$$N_{\text{params}}^{\text{cat}} = 0. \tag{3.27}$$

GRU Submodule

The GRU (gated recurrent unit) submodule is implemented using the PyTorch multi-layer GRU[2], which integrates the GRU [9] introduced in section 2.2 on each layer. As a quick reminder, the single layer GRU has the ability to comprehend temporal relations within sequences. Each layer $l \in \left\{1,\ldots,\breve{N}_{\text{layer}}^{\text{gru}}\right\}$ processes an input sequence by iterating through it $t \in \left\{1,\ldots,\breve{N}^{\text{seq}}\right\}$ mapping the current sequence element and the layer's previous hidden state to the current hidden state (see equ. 2.27)

$$\mathcal{F}_l^{\text{gru}} : \left(\mathbb{R}^{N_{\text{in},l}^{\text{gru}}}, [-1,1]^{\breve{N}_{\text{hidden}}^{\text{gru}}}\right) \to [-1,1]^{\breve{N}_{\text{hidden}}^{\text{gru}}}; \quad \left(\underline{x}_t^{(l)}, \underline{h}_{t-1}^{(l)}\right) \mapsto \underline{h}_t^{(l)}. \tag{3.28}$$

Thereby, $\underline{h}_0^{(l)}$ is either initialized with zeros or corresponds to the last computed hidden state from the last inference. In the multi-layer GRU, each layer maintains its own hidden state. The user specifies the number of layers $\breve{N}_{\text{layer}}^{\text{gru}}$ and the hidden size $\breve{N}_{\text{hidden}}^{\text{gru}}$ (i.e., dimensionality) shared by all hidden states. While the first GRU layer inputs the elements of the given input sequence, all subsequent layers input the hidden state from the previous layer subject to dropout

$$\begin{aligned}\mathcal{F}^{\text{gru}} : &\left(\mathbb{R}^{N_{\text{in},1}^{\text{gru}}}, [-1,1]^{\breve{N}_{\text{hidden}}^{\text{gru}}\times\breve{N}_{\text{layer}}^{\text{gru}}}\right) \to [-1,1]^{\breve{N}_{\text{hidden}}^{\text{gru}}}\\ &\left(\underline{x}_t, \underline{h}_{t-1}^{(1)}, \ldots, \underline{h}_{t-1}^{(\breve{N}_{\text{layer}}^{\text{gru}})}\right) \mapsto \underline{h}_t^{(\breve{N}_{\text{layer}}^{\text{gru}})} = \ldots\\ \ldots &\mathcal{F}_{\breve{N}_{\text{layer}}^{\text{gru}}}^{\text{gru}}\left(\underline{\delta}\odot\mathcal{F}_{\breve{N}_{\text{layer}}^{\text{gru}}-1}^{\text{gru}}\left(\underline{\delta}\odot\ldots\mathcal{F}_1^{\text{gru}}\left(\underline{x}_t, \underline{h}_{t-1}^{(1)}\right), \underline{h}_{t-1}^{(\breve{N}_{\text{layer}}^{\text{gru}}-1)}\right), \underline{h}_{t-1}^{(\breve{N}_{\text{layer}}^{\text{gru}})}\right).\end{aligned} \tag{3.29}$$

[2] `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`, accessed on October 5, 2022

Dropout is a measure that prevents ANNs from overfitting to their provided training data. At training, it randomly sets entries of a feature vector to zero while maintaining the vector's signal strength on average in order to force the ANN to learn the extraction of stand-alone features whose informative value is independent of the other extracted features [21]. Mathematically, dropout applied on a hidden state (or feature vector) is calculated with the Hadamard product (denoted with $\odot$) of that hidden state and the vector $\underline{\delta}$ of same dimensionality whose entries, for every calculation, are random variables resampled from a Bernoulli distribution with a user-specified probability

$$P\left(\delta_i = 0\right) = \breve{p}^{\text{gru}}, \quad P\left(\delta_i = \frac{1}{1-\breve{p}^{\text{gru}}}\right) = 1 - \breve{p}^{\text{gru}}. \tag{3.30}$$

At race, the dropout probability is null whereby the dropout is deactivated.

As the l-th GRU layer has $3\breve{N}^{\text{gru}}_{\text{hidden}}(N^{\text{gru}}_{\text{in},l} + \breve{N}^{\text{gru}}_{\text{hidden}} + 2)$ trainable parameters (see equ. 2.30), the total number of trainable parameters of the GRU submodule is

$$N^{\text{gru}}_{\text{params}} = 3\breve{N}^{\text{gru}}_{\text{hidden}}\left((N^{\text{gru}}_{\text{in},1} + \breve{N}^{\text{gru}}_{\text{hidden}} + 2) + (\breve{N}^{\text{gru}}_{\text{layer}} - 1)(2\breve{N}^{\text{gru}}_{\text{hidden}} + 2)\right). \tag{3.31}$$

The GRU submodule operates in many-to-one mode at training or one-to-one mode at race where the length of the input sequences is one. Either way, the GRU submodule maps its input batch (see equ. 3.26) to a batch of last hidden states of the last layer

$$\underline{h}^{(\breve{N}^{\text{gru}}_{\text{layer}})}_{\breve{N}^{\text{seq}},i}, \quad i \in \left\{1, \ldots, \breve{N}^{\text{batch}}\right\}. \tag{3.32}$$

FC Submodule

The FC submodule performs the function of increasing the general complexity of the ANN module. It consists of multiple fully connected layers. Each layer $l \in \left\{1, \ldots, \breve{N}^{\text{fc}}_{\text{layer}}\right\}$ applies an activation, dropout and a biased linear transformation on the input feature vector

$$\mathcal{F}^{\text{fc}}_l : \mathbb{R}^{N^{\text{fc}}_{\text{in},l}} \to \mathbb{R}^{\breve{N}^{\text{fc}}_{\text{width}}}; \quad \underline{x} \mapsto \underline{\underline{A}}^{\text{fc}}_l \cdot \underline{\delta}^{\text{fc}} \odot \overset{\odot}{\breve{f}}^{\text{fc}}(\underline{x}) + \underline{b}^{\text{fc}}_l. \tag{3.33}$$

In the multi-layer FC submodule, the first layer inputs the given input feature vector, whereas all subsequent layers input the output from the previous layer

$$\mathcal{F}^{\text{fc}} : \mathbb{R}^{N^{\text{fc}}_{\text{in},1}} \to \mathbb{R}^{\breve{N}^{\text{fc}}_{\text{width}}}; \quad \underline{x} \mapsto \mathcal{F}^{\text{fc}}_{\breve{N}^{\text{fc}}_{\text{layer}}}\left(\mathcal{F}^{\text{fc}}_{\breve{N}^{\text{fc}}_{\text{layer}}-1}\left(\ldots \mathcal{F}^{\text{fc}}_1(\underline{x})\right)\right). \tag{3.34}$$

For the FC submodule, the user specifies:

- the number of layers $\breve{N}^{\text{fc}}_{\text{layer}}$

- the width $\breve{N}^{\text{fc}}_{\text{width}}$, i.e., output dimensionality shared by all layers
- the activation function $\breve{f}^{\text{fc}} : \mathbb{R} \to \mathbb{R}$ from the non-linear activations implemented in PyTorch[3], which applies element-wise on the input feature vector (denoted with the overset $\odot$)
- the dropout probability $\breve{p}^{\text{fc}} \in [0, 1]$, i.e., the probability to resample an entry of the vector $\underline{\delta}^{\text{fc}}$ with zero

The input dimensionality of a layer adapts to the number of features in the given input vector. For the first layer, it adapts to the feature vector forwarded to the FC submodule $N^{\text{fc}}_{\text{in},1} = \dim(\underline{x})$. For all subsequent layers $l \geq 2$, it adapts to the width of the FC submodule $N^{\text{fc}}_{\text{in},l} = \breve{N}^{\text{fc}}_{\text{width}}$. The biased linear transformation of the l-th layer consists of the multiplication with the matrix of trainable weights and the addition of the vector of trainable biases

$$\underline{\underline{A}}^{\text{fc}}_l \in \mathbb{R}^{N^{\text{fc}}_{\text{in},l} \times \breve{N}^{\text{fc}}_{\text{width}}}, \quad \underline{b}^{\text{fc}}_l \in \mathbb{R}^{\breve{N}^{\text{fc}}_{\text{width}}}. \tag{3.35}$$

As a single layer therewith has $\left(N^{\text{fc}}_{\text{in},l} + 1\right) \breve{N}^{\text{fc}}_{\text{width}}$ trainable parameters, the total number of trainable parameters of the FC submodule is

$$N^{\text{fc}}_{\text{params}} = \left(N^{\text{fc}}_{\text{in},1} + 1 + \left(\breve{N}^{\text{fc}}_{\text{layer}} - 1\right)\left(\breve{N}^{\text{fc}}_{\text{width}} + 1\right)\right)\breve{N}^{\text{fc}}_{\text{width}}. \tag{3.36}$$

The FC submodule maps the batch of hidden states outputted by the GRU submodule to the batch of feature vectors

$$\mathcal{F}^{\text{fc}}\left(\underline{h}\right)_i, \quad i \in \left\{1, \ldots, \breve{N}^{\text{batch}}\right\}. \tag{3.37}$$

HEAD Submodule

The mandatory HEAD submodule performs the function of mapping to the final output of the ANN module. Depending on the user's selection, the final output is either a navigation decision or a control command. A navigation decision

$$(\tilde{v}^{\text{d}}_{\text{des}}, {}_{\mathbf{I}}\underline{p}^{\text{wp}}) \tag{3.38}$$

comprises a normalized desired speed $\tilde{v}^{\text{d}}_{\text{des}} \in [0, 1]$ of the drone and a waypoint ${}_{\mathbf{I}}\underline{p}^{\text{wp}} \in [\text{-}1, 1]$ in the image reference system (see fig. 3.1b). A control command

$$({}_{\mathbf{L}}\underline{\omega}^{\text{d}}_{\text{des}}, {}_{\mathbf{L}}\underline{\dot{\omega}}^{\text{d}}_{\text{des}}, c^{\text{d}}_{\text{des}}) \tag{3.39}$$

comprises the desired angular velocity ${}_{\mathbf{L}}\underline{\omega}^{\text{d}}_{\text{des}}$ and acceleration ${}_{\mathbf{L}}\underline{\dot{\omega}}^{\text{d}}_{\text{des}}$ of the drone in the local reference system (see fig. 3.1a) as well as the desired collective thrust $c^{\text{d}}_{\text{des}}$ of the

[3] `https://pytorch.org/docs/stable/nn.html`, accessed on October 5, 2022

drone's rotors. In the limited scope of this master's thesis, the control command output option, which is a shortcut to the position controller output (see section 3.4), is only partly implemented and not examined in the experiments in section 4.

The head submodule corresponds to a single layer of the FC submodule without dropout (see equ. 3.33) as it applies an activation and a biased linear transformation on the input feature vector

$$\mathcal{F}^{\text{head}}: \mathbb{R}^{N_{\text{in}}^{\text{head}}} \to \mathbb{R}^{N_{\text{out}}^{\text{head}}}; \quad \underline{x} \mapsto \underline{\underline{A}}^{\text{head}} \cdot \overset{\odot}{\breve{f}}^{\text{head}}(\underline{x}) + \underline{b}^{\text{head}}. \tag{3.40}$$

For the HEAD submodule, the user selects the activation function $\breve{f}^{\text{fc}}: \mathbb{R} \to \mathbb{R}$ from the non-linear activations implemented in PyTorch which applies element-wise on the input feature vector (denoted with the overset $\odot$). The input dimensionality adapts to the number of features of the given input vector

$$N_{\text{in}}^{\text{head}} = \dim(\underline{x}), \tag{3.41}$$

whereas the output dimensionality adapts to the user-selected output option

$$N_{\text{out}}^{\text{head}} = \begin{cases} 3, & \text{if navigation decision} \\ 7, & \text{if control command.} \end{cases} \tag{3.42}$$

The total number of trainable parameters of the HEAD submodule is

$$N_{\text{params}}^{\text{head}} = \left(N_{\text{in}}^{\text{head}} + 1\right) N_{\text{out}}^{\text{head}}. \tag{3.43}$$

3.3 Planning Module

The planning module performs the task of path planning within the autonomous navigation method. At the user-specified main frequency $\breve{f}^{\text{main}}$, the planning module samples states from its local trajectory and forwards them as reference to the control module. Every $\breve{N}^{\text{plan}}$-th (user-specified) iteration, the planning module re-computes its local trajectory on the basis of its input, i.e., the latest navigation decision and the latest drone state estimate.

The latest navigation decision stems from either the ANN module or, if it has intervened at training data generation, the expert system. A navigation decision comprises the normalized desired speed and the waypoint in the image reference system

$$(\tilde{v}_{\text{des}}^{\text{d}},\ {}_{\mathbf{I}}\underline{p}^{\text{wp}}). \tag{3.44}$$

The latest drone state estimate stems from the state estimation system. In simulation, the estimate may correspond to the ground-truth state. A drone state estimate includes position, velocity and acceleration with respect to the global reference system

$${}_{\mathbf{G}}\underline{p}^{\text{d}},\ {}_{\mathbf{G}}\underline{v}^{\text{d}},\ {}_{\mathbf{G}}\underline{a}^{\text{d}}. \tag{3.45}$$

At a fraction of the main frequency, i.e., $\breve{f}^{\text{main}}/\breve{N}^{\text{plan}}$, the planning module takes the following 5 steps to re-compute its local trajectory.

1. Compute the desired speed

$$v_{\text{des}}^{\text{d}} = \max\left(\breve{v}_{\text{min}}^{\text{d}},\ \breve{v}_{\text{max}}^{\text{d}} \cdot \tilde{v}_{\text{des}}^{\text{d}}\right). \tag{3.46}$$

   The normalized, desired speed $\tilde{v}_{\text{des}}^{\text{d}} \in [0, 1]$ of the navigation decision is rescaled by its upper bound, the user-specified drone's maximum speed $\breve{v}_{\text{max}}^{\text{d}}$. The user-specified drone's minimum speed $\breve{v}_{\text{min}}^{\text{d}}$ lower-bounds the desired speed.

2. Compute the drone's distance to the waypoint

$$d^{\text{d-wp}} = \max\left(\breve{d}_{\text{min}}^{\text{d-wp}},\ \min\left(v_{\text{des}}^{\text{d}} \cdot \breve{t}_{\Delta}^{\text{d-wp}},\ \breve{d}_{\text{max}}^{\text{d-wp}}\right)\right). \tag{3.47}$$

   The desired speed $v_{\text{des}}^{\text{d}}$ is integrated over the user-specified duration $\breve{t}_{\Delta}^{\text{d-wp}}$. The result is bounded to the interval spanned by the user-specified minimum $\breve{d}_{\text{min}}^{\text{d-wp}}$ and maximum $\breve{d}_{\text{max}}^{\text{d-wp}}$ distance.

3. Compute the waypoint with respect to the global reference system

$${}_{\mathbf{G}}\underline{p}^{\text{wp}} = T_{\mathbf{GI}}\left(d^{\text{d-wp}},\ {}_{\mathbf{I}}\underline{p}^{\text{wp}}\right). \tag{3.48}$$

   The transformation $T_{\mathbf{GI}}$ (see equ. 3.14) back-projects the waypoint ${}_{\mathbf{I}}\underline{p}^{\text{wp}}$ of the navigation decision from the 2D image to the 3D global reference system. Thereby, the drone's distance $d^{\text{d-wp}}$ to the waypoint constitutes the back-projection length.

4. Set the starting time of the local trajectory to the current time

$$t_0^{\text{lt}} = t \tag{3.49}$$

   and compute the duration of the local trajectory

$$t_{\Delta}^{\text{lt}} = \frac{d^{\text{d-wp}}}{\min\left(v_{\text{des}}^{\text{d}},\ \left\|{}_{\mathbf{G}}\underline{v}^{\text{d}}\right\|_2 + \breve{v}_{\Delta}^{\text{d}}\right)}. \tag{3.50}$$

   The drone's distance $d^{\text{d-wp}}$ to the waypoint is divided by the slowest of either the desired speed $v_{\text{des}}^{\text{d}}$ or the latest drone speed estimate $\left\|{}_{\mathbf{G}}\underline{v}^{\text{d}}\right\|_2$ plus a user-specified speed increment $\breve{v}_{\Delta}^{\text{d}}$. By relating the desired to the estimated speed, excessive speed increases potentially violating the drone's dynamic limitations can be prevented.

5. Compute the local trajectory

$$ {}_{\mathbf{G}}\underline{p}^{\mathrm{lt}} : \left[0, t_{\Delta}^{\mathrm{lt}}\right] \to \mathbb{R}^3; \quad t \mapsto {}_{\mathbf{G}}\underline{p}^{\mathrm{lt}}(t) \tag{3.51} $$

starting in the latest drone state estimate ${}_{\mathbf{G}}\underline{p}^{\mathrm{d}}$, ${}_{\mathbf{G}}\underline{v}^{\mathrm{d}}$, ${}_{\mathbf{G}}\underline{a}^{\mathrm{d}}$ and ending in the global waypoint ${}_{\mathbf{G}}\underline{p}^{\mathrm{wp}}$ with unconstrained velocity and acceleration. The implementation[4] of the algorithm of Mueller et al. [43] is deployed to find the polynomial trajectory with minimum jerk (third time derivative of position) by solving the optimization problem

$$ \begin{aligned} & \min \int_0^{t_{\Delta}^{\mathrm{lt}}} \left\| {}_{\mathbf{G}}\dddot{\underline{p}}^{\mathrm{lt}}(t) \right\|_2^2 \mathrm{d}t \\ \text{s.t.} \quad & {}_{\mathbf{G}}\underline{p}^{\mathrm{lt}}(0) = {}_{\mathbf{G}}\underline{p}^{\mathrm{d}} && {}_{\mathbf{G}}\underline{p}^{\mathrm{lt}}(t_{\Delta}^{\mathrm{lt}}) = {}_{\mathbf{G}}\underline{p}^{\mathrm{wp}} \\ & {}_{\mathbf{G}}\dot{\underline{p}}^{\mathrm{lt}}(0) = {}_{\mathbf{G}}\underline{v}^{\mathrm{d}} && {}_{\mathbf{G}}\dot{\underline{p}}^{\mathrm{lt}}(t_{\Delta}^{\mathrm{lt}}) \text{ free} \\ & {}_{\mathbf{G}}\ddot{\underline{p}}^{\mathrm{lt}}(0) = {}_{\mathbf{G}}\underline{a}^{\mathrm{d}} && {}_{\mathbf{G}}\ddot{\underline{p}}^{\mathrm{lt}}(t_{\Delta}^{\mathrm{lt}}) \text{ free.} \end{aligned} \tag{3.52} $$

The drone's dynamic limitations are only taken into account in subsequent feasibility checks and are exempt from the above optimization problem. This allows the algorithm to solve the optimization problem in closed form which is characterized by low computational effort. The algorithm therewith qualifies to run at the relatively high frequencies required by the autonomous navigation method.

At the main frequency $\breve{f}^{\mathrm{main}}$, the planning module takes the following 2 steps to sample a reference state from its local trajectory.

1. Compute the time point of the reference state

$$ t^{\mathrm{lt}} = t - t_0^{\mathrm{lt}} + 1/\breve{f}^{\mathrm{main}}. \tag{3.53} $$

The actual time t is related to the starting time t_0^{lt} of the local trajectory, whose time domain starts at zero. The addition of the main period $1/\breve{f}^{\mathrm{main}}$ ensures that the sampled reference state remains prospective at all times.

2. Sample the current reference state from the local trajectory

$$ \begin{aligned} {}_{\mathbf{G}}\underline{p}^{\mathrm{ref}} &= {}_{\mathbf{G}}\underline{p}^{\mathrm{lt}}(t^{\mathrm{lt}}) \\ {}_{\mathbf{G}}\underline{v}^{\mathrm{ref}} &= {}_{\mathbf{G}}\dot{\underline{p}}^{\mathrm{lt}}(t^{\mathrm{lt}}) \\ {}_{\mathbf{G}}\underline{a}^{\mathrm{ref}} &= {}_{\mathbf{G}}\ddot{\underline{p}}^{\mathrm{lt}}(t^{\mathrm{lt}}) \\ {}_{\mathbf{G}}\underline{j}^{\mathrm{ref}} &= {}_{\mathbf{G}}\dddot{\underline{p}}^{\mathrm{lt}}(t^{\mathrm{lt}}) \\ \phi_z^{\mathrm{ref}} &= \operatorname{atan2}\left({}_{\mathbf{G}}v_y^{\mathrm{ref}}, {}_{\mathbf{G}}v_x^{\mathrm{ref}}\right). \end{aligned} \tag{3.54} $$

The reference yaw ϕ_z^{ref} is set so that the drone and therewith its onboard camera point in the direction of flight movement.

[4] `https://github.com/markwmuller/RapidQuadrocopterTrajectories`, accessed on October 5, 2022

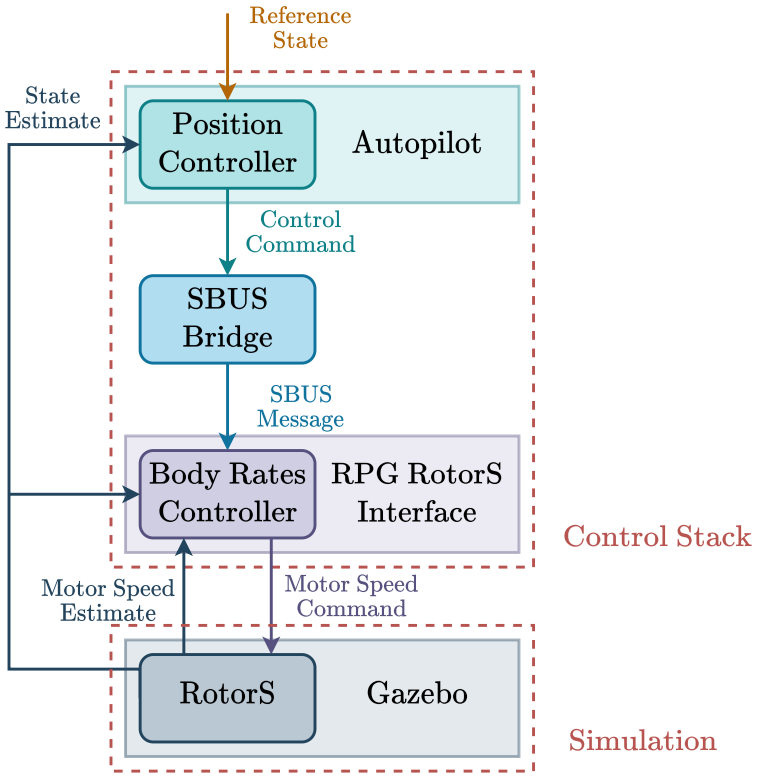


Figure 3.3: Control stack in simulation

3.4 Control Stack

Within the autonomous navigation method, the control stack takes on the task of flying the drone as planned. To do this, the control stack generates the inputs for the motors attached to the drone's rotors, which consequently track the latest reference state (equ. 3.54) from the planning module. In the simulations of this thesis, the control stack is realized with the RPG Quadrotor Control[5] implementation (see fig. 3.3). Since this thesis centers on the reasoning aspect of autonomous navigation, only an overview of the deployed control stack is presented here. The reader may consult the provided references for more details on the control.

The RPG Quadrotor Control implementation, which includes the autopilot, the SBUS bridge and and the RPG RotorS interface, basically executes two feedback control loops in a cascade. The autopilot integrates the position controller that runs the control algorithm of Faessler at al. proposed in [15]. Based on the latest reference state and the fed back drone state estimates, the position controller generates high-level

[5] `https://github.com/uzh-rpg/rpg_quadrotor_control`, accessed on October 5, 2022

control commands. A control command comprises the collective thrust of the drone's rotors as well as the drone's angular velocity and acceleration. The SBUS bridge converts each incoming control command into an SBUS message and forwards this message to the RPG RotorS interface. The RPG RotorS interface integrates the body-rate controller that runs the control algorithm of Faessler at al. proposed in [14]. Based on the latest high-level control command as well as the fed back drone state and motor speed estimates, the body-rate controller generates low-level motor speed commands. These commands are forwarded to RotorS for execution. RotorS, developed by Furrer et al. [17], is plugged into the Gazebo[6] simulator to model the drone's physics and to provide the controllers with drone state and motor speed estimates.

In real-world, the drone's flight controller would replace the RPG RotorS interface in order to generate hardware-specific low-level motor commands based on the latest SBUS message from the SBUS bridge.

3.5 Expert System

In the context of machine learning, an expert system is a program that imitates a human expert in order to solve a problem. It comprises a knowledge base, which stores known facts and rules, and an inference engine, which infers new facts by applying the rules to the known facts [26].

Problem
The interactive expert system demonstrates the desired navigation decision-making in the dataset aggregation imitation learning process of the ANN module. The training dataset is extended with new samples while the drone runs the autonomous navigation method to fly through a racetrack. The expert system checks the latest navigation decision made by the yet partially trained ANN module. If it does not meet certain requirements, the expert system intervenes with its own navigation decision. This, first, keeps the drone on course and, second, triggers the generation of a new training sample labeled with the expert system's navigation decision.

Knowledge base
While the ANN module infers navigation decisions from onboard sensor data, the expert system makes navigation decisions based on its knowledge which includes the following known facts (**F***) and rules (**R***).

F1 The planning module's waypoint

$$ {}_{\mathbf{G}}\underline{p}^{\text{ann}}_{\text{wp}} \tag{3.55} $$

[6] `https://gazebosim.org/home`, accessed on October 5, 2022

with respect to the global reference system (see equ. 3.48) that was computed based on the ANN module's latest navigation decision (see equ. 3.38).

F2 The drone's latest position and quaternion orientation estimate, which are provided by the drone's state estimation system and may correspond to ground truth in the simulation

$$ {}_{\mathbf{G}}\underline{p}^{\mathrm{d}}, \quad {}_{\mathbf{G}}\underline{q}^{\mathrm{d}}. \tag{3.56} $$

F3 The center points of the gates of the racetrack

$$ \left({}_{\mathbf{G}}\underline{p}_i^{\mathrm{gate}} \right)_{i \in \{0,\ldots,N^{\mathrm{gate}}-1\}} \tag{3.57} $$

and the initial index to the currently targeted gate to be passed next

$$ i_{\mathrm{target}}^{\mathrm{gate}} \in \{0, ..., N^{\mathrm{gate}} - 1\}. \tag{3.58} $$

R1 Compute the global trajectory of the current racetrack

$$ {}_{\mathbf{G}}\underline{p}^{\mathrm{gt}} : \left[t_0^{\mathrm{gate}}, t_{N^{\mathrm{gate}}}^{\mathrm{gate}} \right] \to \mathbb{R}^3; \quad t \mapsto {}_{\mathbf{G}}\underline{p}^{\mathrm{gt}}(t). \tag{3.59} $$

The algorithm of Mellinger and Kumar [39] finds the minimum snap (fourth time derivative of position) spline trajectory

$$ {}_{\mathbf{G}}\underline{p}^{\mathrm{gt}}(t) = \sum_{i=0}^{N^{\mathrm{gate}}-1} \begin{cases} {}_{\mathbf{G}}\underline{p}_i^{\mathrm{gt}}(t), & t \in \left[t_i^{\mathrm{gate}}, t_{i+1}^{\mathrm{gate}} \right] \\ 0, & \text{else} \end{cases} \tag{3.60} $$

that, traverses through all gate center points (**F3**), each at its corresponding gate time t_i^{gate}, and reconnects to itself at $t = t_{N^{\mathrm{gate}}}^{\mathrm{gate}}$ at gate $i = 0$. The entries of the pieces ${}_{\mathbf{G}}\underline{p}_i^{\mathrm{gt}}(t)$ of the spline are polynomials. The user specifies the polynomial order $\breve{N}_{\mathrm{poly}}^{\mathrm{gt}}$ of the pieces and the continuity order $\breve{N}_{\mathrm{cont}}^{\mathrm{gt}}$ of the spline. However, since the goal is to minimize snap, it is required that $\breve{N}_{\mathrm{poly}}^{\mathrm{gt}} \geq \breve{N}_{\mathrm{cont}}^{\mathrm{gt}} \geq 4$. The algorithm performs the following two-step iterative optimization.

First, the optimal polynomial coefficients of the spline pieces are found for fixed gate arrival times t_i^{gate} by solving the optimization problem

$$ \begin{aligned} &\underset{{}_{\mathbf{G}}\underline{p}^{\mathrm{gt}}}{\operatorname{argmin}} \int_{t_0^{\mathrm{gate}}}^{t_{N^{\mathrm{gate}}}^{\mathrm{gate}}} \left\| {}_{\mathbf{G}}\dddot{\underline{p}}^{\mathrm{gt}}(t) \right\|_2^2 \mathrm{d}t \\ &\text{s.t.} \quad {}_{\mathbf{G}}\underline{p}^{\mathrm{gt}}\left(t_i^{\mathrm{gate}}\right) = {}_{\mathbf{G}}\underline{p}_i^{\mathrm{gate}}, \qquad \frac{\mathrm{d}^j {}_{\mathbf{G}}\underline{p}^{\mathrm{gt}}}{\mathrm{d}t^j}(t_i^{\mathrm{gate}}) \text{ defined}, \\ &\qquad i \in \{0, ..., N^{\mathrm{gate}}\}, \qquad j \in \left\{1, ..., \breve{N}_{\mathrm{cont}}^{\mathrm{gt}}\right\}. \end{aligned} \tag{3.61} $$

Note that, as the spline is closed, the first and last gate equate ${}_{\mathbf{G}}\underline{p}^{\text{gate}}_{N^{\text{gate}}} = {}_{\mathbf{G}}\underline{p}^{\text{gate}}_{0}$. Morover, the gate times of the very first iteration are approximated with the distances between the gate center points divided by the user-specified maximum speed $\breve{v}^{\text{gt}}_{\text{max}}$ of the trajectory. The above optimization problem is temporally and spatially de-dimensionalized to increase numeric stability and reformulated as quadratic program, which is solved with the Gurobi[7] optimizer.

Second, the polynomial coefficients of the spline pieces are fixed and the inner gate times t^{gate}_i are optimized relatively to each other. The corresponding optimization problem

$$\begin{aligned} &\underset{t^{\text{gate}}_i}{\operatorname{argmin}} \int_{t^{\text{gate}}_0}^{t^{\text{gate}}_{N^{\text{gate}}}} \left\| {}_{\mathbf{G}}\dddot{\underline{p}}^{\text{gt}}(t) \right\|_2^2 \mathrm{d}t \\ &\text{s.t.} \quad t^{\text{gate}}_i < t^{\text{gate}}_{i+1}, \qquad t^{\text{gate}}_0,\ t^{\text{gate}}_{N^{\text{gate}}} \text{ fixed}, \qquad i \in \{0, ..., N^{\text{gate}} - 1\} \end{aligned} \tag{3.62}$$

is solved by gradient descent with backtracking line search.

The two optimization steps are executed iteratively until the cost of the first optimization problem converges. Then, the trajectory is temporally and spatially re-dimensionalized and temporally scaled to adhere to the user-specified maximum values in terms of speed $\breve{v}^{\text{gt}}_{\text{max}}$, thrust $\breve{a}^{\text{gt}}_{\text{max}}$ and roll-pitch rate $\breve{\omega}^{\text{gt}}_{\text{max}}$ along the trajectory. For later use, the expert system samples the positions and speeds of the global trajectory

$$\left({}_{\mathbf{G}}\underline{p}^{\text{gt}}_i \right)_{i \in \{0,\dots,N^{\text{gt}}-1\}}, \quad \left({}_{\mathbf{G}}v^{\text{gt}}_i \right)_{i \in \{0,\dots,N^{\text{gt}}-1\}} \tag{3.63}$$

with ${}_{\mathbf{G}}v^{\text{gt}}_i = \left\| {}_{\mathbf{G}}\dot{\underline{p}}^{\text{gt}}_i \right\|_2$. The sampling occurs at the user-specified frequency $\breve{f}^{\text{gt}}$, which results in $N^{\text{gt}} = \breve{f}^{\text{gt}} \cdot \left(t^{\text{gate}}_{N^{\text{gate}}} - t^{\text{gate}}_0 \right)$ samples.

R2 If the drone is closer to the currently targeted gate than a user-specified distance

$$\left\| {}_{\mathbf{G}}\underline{p}^{\text{gate}}_{i^{\text{gate}}_{\text{target}}} - {}_{\mathbf{G}}\underline{p}^{\text{d}} \right\|_2 < \breve{d}^{\text{d-gate}}, \tag{3.64}$$

increment the index to the currently targeted gate

$$i^{\text{gate}} \leftarrow (i^{\text{gate}} + 1) \bmod N^{\text{gate}}. \tag{3.65}$$

R3 If the planning module's waypoint (**F1**) computed based on the ANN module's latest navigation decision, is more distant from the global trajectory (**R1**) than a user-specified margin scaled by the user-specified drone's maximum speed, i.e.,

$$\underset{i \in \{0,\dots,N^{\text{gt}}-1\}}{\operatorname{argmin}} \left\| {}_{\mathbf{G}}\underline{p}^{\text{ann}}_{\text{wp}} - {}_{\mathbf{G}}\underline{p}^{\text{gt}}_i \right\|_2 > \breve{d}^{\text{wp-gt}}_{\text{max}} \cdot \frac{\breve{v}^{\text{d}}_{\text{max}} + 1}{5}, \tag{3.66}$$

[7] `https://www.gurobi.com/`, accessed on October 5, 2022

the expert system is required to intervene with its own navigation decision.

R4 Update the index $i_{\text{proj}}^{\text{gt}} \in \{0, ..., N^{\text{gt}}-1\}$ to the projection state, i.e., the state of the global trajectory onto which the drone's latest position estimate is projected, with the following iterative method. Figure 3.4 schematically illustrates the method with a 2D example.

1. Compute the index to the previous state of the global trajectory, by decrementing the index to the projection state

$$i_{\text{prev}}^{\text{gt}} = (i_{\text{proj}}^{\text{gt}} - 1 + N^{\text{gt}}) \bmod N^{\text{gt}}. \tag{3.67}$$

2. Starting from the previous state, compute the vector to the projection state

$${}_{\mathbf{G}}\underline{a} = {}_{\mathbf{G}}\underline{p}_{i_{\text{proj}}^{\text{gt}}}^{\text{gt}} - {}_{\mathbf{G}}\underline{p}_{i_{\text{prev}}^{\text{gt}}}^{\text{gt}} \tag{3.68}$$

and the vector to the current drone position

$${}_{\mathbf{G}}\underline{b} = {}_{\mathbf{G}}\underline{p}^{\text{d}} - {}_{\mathbf{G}}\underline{p}_{i_{\text{prev}}^{\text{gt}}}^{\text{gt}}. \tag{3.69}$$

3. If the scalar product of the vectors ${}_{\mathbf{G}}\underline{a}$ and ${}_{\mathbf{G}}\underline{b}$, both normalized by the length of ${}_{\mathbf{G}}\underline{a}$, is less than 1

$$\frac{{}_{\mathbf{G}}\underline{a} \cdot {}_{\mathbf{G}}\underline{b}}{{}_{\mathbf{G}}\underline{a} \cdot {}_{\mathbf{G}}\underline{a}} < 1, \tag{3.70}$$

go to the next step. Else, increment the index to the projection state

$$i_{\text{proj}}^{\text{gt}} \leftarrow (i_{\text{proj}}^{\text{gt}} + 1) \bmod N^{\text{gt}} \tag{3.71}$$

and go back to step 1.

4. If the drone is within a user-specified distance to the projection state

$$\left\| {}_{\mathbf{G}}\underline{p}^{\text{d}} - {}_{\mathbf{G}}\underline{p}_{i_{\text{proj}}^{\text{gt}}}^{\text{gt}} \right\|_2 \leq \breve{d}^{\text{d-proj}}, \tag{3.72}$$

the index $i_{\text{proj}}^{\text{gt}}$ to the projection state is found. Else, set the index to the state of the global trajectory which has the minimum distance to the current drone position

$$\underset{i_{\text{proj}}^{\text{gt}}}{\operatorname{argmin}} \left\| {}_{\mathbf{G}}\underline{p}^{\text{d}} - {}_{\mathbf{G}}\underline{p}_{i_{\text{proj}}^{\text{gt}}}^{\text{gt}} \right\|_2. \tag{3.73}$$

Due to this step, the expert system does not require to know the initial index to the projection state.

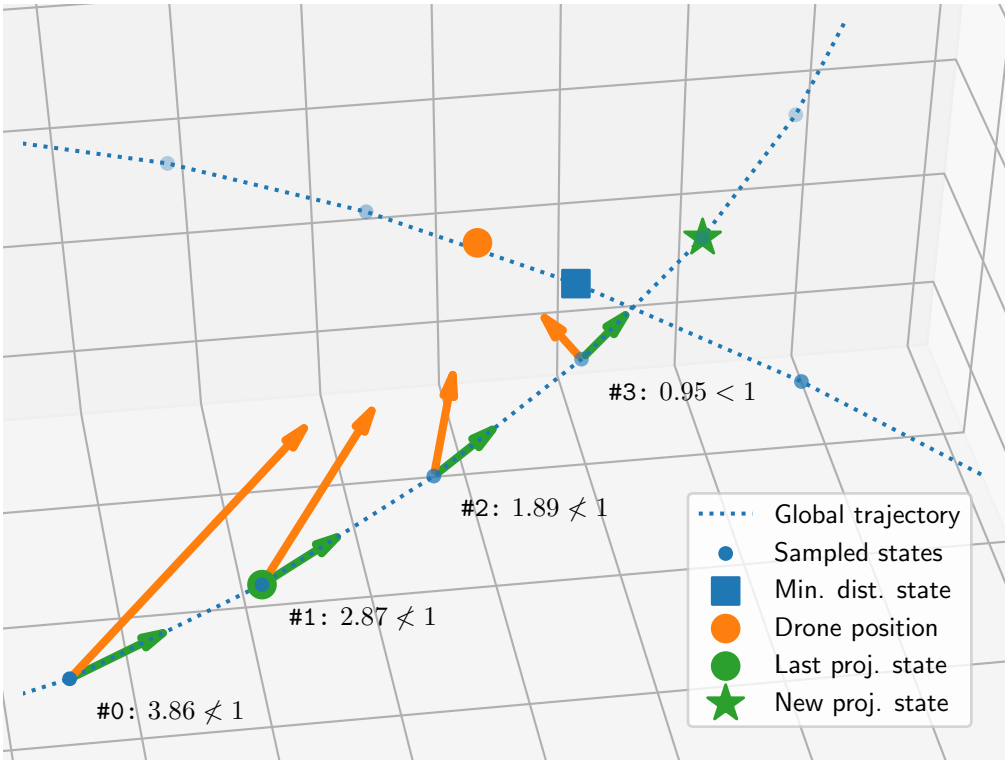


Figure 3.4: Update of the projection state index (**R4**). Known are: the positions (blue points) sampled from the global trajectory (blue dotted line), the last index to the projection state (green circle) and the current position of the drone (orange circle). At an iteration, the vector from the previous to the projection state (green arrows, equ. 3.68) and the vector from the previous to the drone position (orange arrows, equ. 3.69) are computed. Then, the normalized dot product criterion (annotations, equ. 3.70) is checked. For iteration #0-2, the criterion is not met. Thus, the index to the projection state is incremented and another iteration is started. At iteration #3 the criterion is met and the new index to the projection state (green star) is identified (assuming the distance criterion (equ. 3.72) is also met). Note that finding the index to the projection state only with minimum distance (equ. 3.73) would have failed here, since the so indexed state (blue square) belongs to a later or earlier part of the global trajectory which only intersects the current part.

R5 Update the index $i_v^{\text{gt}} \in \{0, ..., N^{\text{gt}} - 1\}$ to the speed state, i.e., the state of the global trajectory that is the reference for the normalized speed $\tilde{v}_{\text{des}}^{\text{exp}}$ component of the expert system's navigation decision, by finding the first state of the global trajectory that follows the projection state with a specific distance.

1. Initialize the searched index with the index to the projection state
$$i_v^{\text{gt}} = i_{\text{proj}}^{\text{gt}}. \tag{3.74}$$
2. Increment the searched index
$$i_v^{\text{gt}} \leftarrow (i_v^{\text{gt}} + 1) \bmod N^{\text{gt}}. \tag{3.75}$$
3. If the speed state is further from the projection state than a user-specified distance
$$\left\| {}_{\mathbf{G}}\underline{p}_{i_v^{\text{gt}}}^{\text{gt}} - {}_{\mathbf{G}}\underline{p}_{i_{\text{proj}}^{\text{gt}}}^{\text{gt}} \right\|_2 > \breve{d}^{\text{proj-}v}. \tag{3.76}$$
the searched index is found. Else, go back to step 2.

R6 Update the index $i_{\text{wp}}^{\text{gt}} \in \{0, ..., N^{\text{gt}} - 1\}$ to the waypoint state, i.e., the state of the global trajectory that is the reference for the image waypoint ${}_{\mathbf{I}}\underline{p}_{\text{wp}}^{\text{exp}}$ component of the expert system's navigation decision, by finding the first state of the global trajectory that follows the projection state with a distance to be computed.

1. Set the distance from the projection to the waypoint state to the distance from the drone to the closer of either the currently or lastly targeted gate. However, a user-specified distance constitutes the lower limit
$$d^{\text{proj-wp}} = \max\left(\breve{d}_{\text{min}}^{\text{proj-wp}},\ \underset{i}{\text{argmin}} \left\| {}_{\mathbf{G}}\underline{p}_i^{\text{gate}} - {}_{\mathbf{G}}\underline{p}^{\text{d}} \right\|_2\right), \tag{3.77}$$
$$i \in \left\{ i_{\text{target}}^{\text{gate}}, (i_{\text{target}}^{\text{gate}} - 1 + N^{\text{gate}}) \bmod N^{\text{gate}} \right\}. \tag{3.78}$$
2. Initialize the searched index with the index to the projection state
$$i_{\text{wp}}^{\text{gt}} = i_{\text{proj}}^{\text{gt}}. \tag{3.79}$$
3. Increment the searched index
$$i_{\text{wp}}^{\text{gt}} \leftarrow (i_{\text{wp}}^{\text{gt}} + 1) \bmod N^{\text{gt}}. \tag{3.80}$$
4. If the waypoint state is further from the projection state than the distance computed in step 1
$$\left\| {}_{\mathbf{G}}\underline{p}_{i_{\text{wp}}^{\text{gt}}}^{\text{gt}} - {}_{\mathbf{G}}\underline{p}_{i_{\text{proj}}^{\text{gt}}}^{\text{gt}} \right\|_2 > d^{\text{proj-wp}}, \tag{3.81}$$
the searched index is found. Else, go back to step 3.

R7 Compute the normalized speed component of the expert system's navigation decision as the sampled speed of the speed state normalized by the maximum speed of the global trajectory

$$\tilde{v}_{\text{des}}^{\text{exp}} = \frac{{}_{\mathbf{G}}v_{i_v^{\text{gt}}}^{\text{gt}}}{\underset{i \in \{0,\dots,N^{\text{gt}}-1\}}{\operatorname{argmax}} \left\| {}_{\mathbf{G}}v_i^{\text{gt}} \right\|_2} \in [0,1]. \tag{3.82}$$

R8 Compute the image waypoint component of the expert system's navigation decision by applying the transformation from the global to the image reference system (see equ. 3.14) on the sampled position of the waypoint state

$${}_{\mathbf{I}}\underline{p}_{\text{wp}}^{\text{exp}} = T_{\mathbf{IG}} \left({}_{\mathbf{I}}\underline{p}_{i_{\text{wp}}^{\text{gt}}}^{\text{gt}} \right). \tag{3.83}$$

Inference Engine

The inference engine of the expert system is only activated during training data generation and runs as follows.

Before the drone starts to fly, the inference engine pre-computes the global trajectory (**R1**) and samples the position and speeds. During the flight, it constantly updates the currently targeted gate index (**R2**). Whenever the planning module has computed a global waypoint on the basis of the latest ANN navigation decision, the inference engine checks whether it must intervene (**R3**). If so, the engine updates its indices to relevant states of the global trajectory (**R4-6**) and makes its own navigation decision (**R7-8**). Finally, the engine sends its navigation decision to the planning module for processing.

Chapter 4

Experiments in Simulation

4.1 Simulation Setup

The experiments in this thesis are conducted in a drone racing simulation, which includes the environment, the racetrack consisting of race gates and the drone. The implementation of the simulation (see fig. 4.1) is separated into physics modeling and image rendering.

For a physics modeling of high accuracy, the Gazebo[1] simulator with the RotorS [17] plugin is deployed. The modeling includes the dynamics of the drone under the actuation with inputted motor speed commands and possible collisions of the drone with the racetrack gates. Further, the RotorS plugin provides the drone state estimate, the motor speeds estimate and the data from the onboard IMU as output.

For an almost photo-realistic image rendering, the Flightmare [57] simulator is de-

[1] `https://gazebosim.org/home`, accessed on October 5, 2022

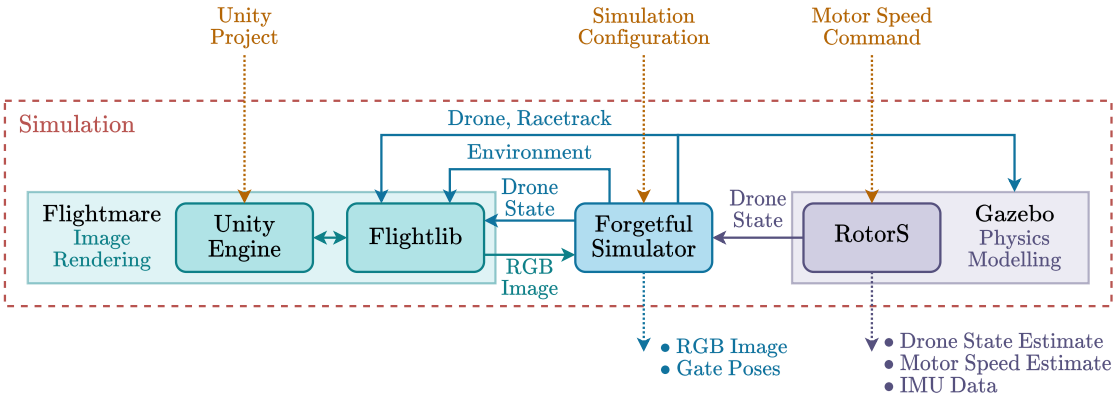
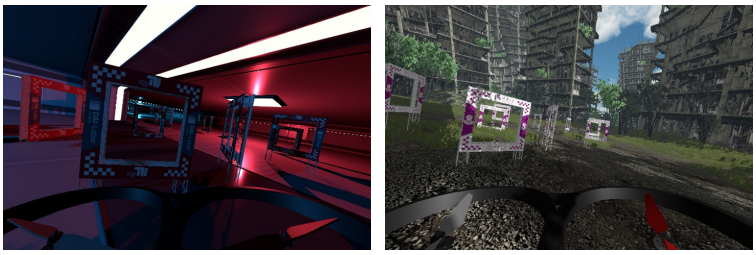


Figure 4.1: Implementation concept of the simulation



(a) Spaceship Interior (b) Destroyed City

(c) Industrial Site (d) Polygon City (e) Desert Mountain

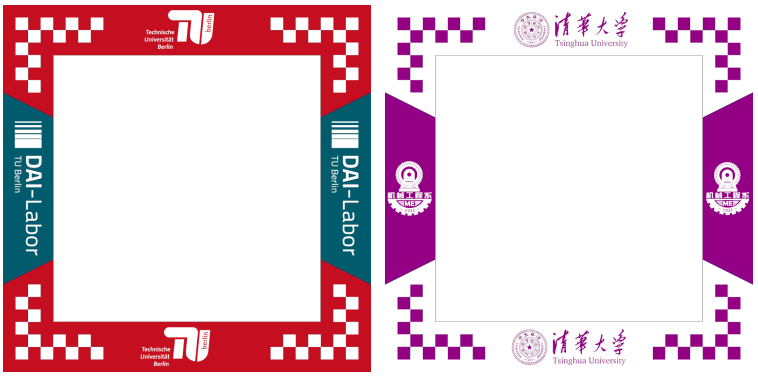
Figure 4.2: Scenes available in simulation

ployed. Upon request, the Flightlib interface updates the drone pose within the Unity[2] Engine and fetches an RGB image from the drone's onboard camera. Before running the simulation, the Unity Engine is built from a Unity project based on the RPG Flightmare Unity Project[3]. The Unity project of this thesis entails five scenes named spaceship interior[4], destroyed city[5], industrial site[6], polygon city[7] and desert mountain[8] (see fig. 4.2). The scenes base on assets from the Unity Asset Store[9]. Within each scene, there are three sites (A, B, C) to place a racetrack. For the racetrack, two different gate types are provided: the first with TU Berlin/DAI-Labor logos and the second with Tsinghua University/DME logos (see fig. 4.3).

The Forgetful Simulator node takes on two tasks. First, it synchronizes the drone state in the Flightmare simulator with the ground-truth state in the Gazebo simulator and provides the RGB images fetched from the Flightmare simulator as output. Second, the node sets up the simulation according to the inputted simulation configuration.

A simulation configuration includes the environment and the racetrack configura-

[2] `https://unity.com/`, accessed on October 5, 2022
[3] `https://github.com/uzh-rpg/flightmare_unity`, accessed on October 5, 2022
[4] based on "3D Free Modular Kit" from the Unity Asset Store
[5] based on "Destroyed City FREE" from the Unity Asset Store
[6] based on "RPG/FPS Game Assets for PC/Mobile (Industrial Set v2.0)" from the Unity Asset Store
[7] based on "CITY package" from the Unity Asset Store
[8] based on "Free Island Collection" from the Unity Asset Store
[9] `https://assetstore.unity.com/`, accessed on October 5, 2022



(a) DAI-Labor at TU Berlin (b) DME at Tsinghua University

Figure 4.3: Race gates available in simulation

tion. The environment configuration specifies the scene and the site. The racetrack configuration specifies the racetrack type, the racetrack generation, the racetrack direction and the racetrack gates. Table 4.1 shows all available options for the simulation configuration. The Forgetful Simulator node stores the gate poses for both deterministic

Table 4.1: Simulation configuration options

Simulation	Environment	Scenes	Spaceship interior, destroyed city, industrial site, polygon city, desert mountain
		Sites	A, B, C
	Racetrack	Types	Figure-8, gap
		Generations	Deterministic, randomized
		Directions	Counterclockwise, clockwise
		Gates	TUB-DAI, THU-DME

and counterclockwise racetrack types (see table 4.2). If specified, these poses are randomized and redirected from counterclockwise to clockwise as illustrated in figure 6.1. The racetrack randomization includes the following steps.

1. Sample the y-position values of the gates #3-6 from the uniform real distribution over the intervals specified in table 4.2. This step only applies to the gap racetrack type, as it explicitly randomizes the gap distance.

2. Shift the gate positions along the x-, y- and z-axis by a value, which is sampled independently for each gate and axis from the uniform real distribution over the user-specified interval $\left[-\breve{d}^{\text{sim}}_{\text{shift,max}}, \breve{d}^{\text{sim}}_{\text{shift,max}}\right]$.

3. Scale the gate positions by a value, which is sampled once for all gates from the uniform real distribution over the user-specified interval $\left[\breve{d}^{\text{sim}}_{\text{shift,min}}, \breve{d}^{\text{sim}}_{\text{shift,max}}\right]$.

4. Twist the gate yaw-orientations by a value, which is sampled independently for each gate from the uniform real distribution over the user-specified interval $\left[-\breve{d}^{\text{sim}}_{\text{twist,max}}, \breve{d}^{\text{sim}}_{\text{twist,max}}\right]$.

After processing the racetrack configuration, the Forgetful Simulator node computes the start position of the drone so that the drone is located between the second last and the last gate and faces towards the last gate. Then, the node loads the environment configuration in the Flightmare Simulator and spawns the drone model and the race gate models of the specified type at the computed poses in both the Flightmare and the Gazebo simulator. Finally, the node outputs the computed gate poses. This information is required to compute the global trajectory of the expert system (see section 3.5) and to update the drone's progress on the racetrack automatically whenever the drone has passed the currently targeted race gate.

Table 4.2: Deterministic gate poses

Racetrack	Gate	x	y	z	yaw
Figure-8	0	-20.45	-8.65	2.0	1.13
	1	-12.55	-11.15	2.0	-1.57
	2	-4.15	-5.35	2.0	-0.60
	3	3.45	4.25	2.0	-0.63
	4	11.95	11.15	2.0	-1.21
	5	21.85	6.85	2.0	0.99
	6	24.25	-1.75	2.0	0.00
	7	19.25	-9.55	2.0	-1.03
	8	10.55	-10.65	2.0	1.53
	9	2.85	-5.95	2.0	0.57
	10	-4.95	4.65	2.0	0.67
	11	-12.95	9.65	2.0	-1.53
	12	-21.05	6.65	2.0	-0.77
	13	-24.25	-1.00	2.0	0.07
Gap	0	-20.45	-8.65	2.0	1.13
	1	-12.55	-11.15	2.0	-1.57
	2	-4.15	-9.35	2.0	-1.0
	3	4.85	[-4.95, -5.95]	2.0	-1.4
	4	16.95	[-2.25, -5.25]	2.0	1.57
	5	16.95	[2.25, 5.25]	2.0	1.57
	6	5.45	[4.45, 5.45]	2.0	1.4
	7	-4.95	7.95	2.0	1.2
	8	-12.95	9.65	2.0	-1.53
	9	-21.05	6.65	2.0	-0.77
	10	-24.25	-1.0	2.0	0.07

4.2 Imitation Learning Process

The ANN module of the autonomous navigation method must first learn to make meaningful navigation decisions. In this thesis, as well as in the baseline work, this task is viewed as an imitation learning problem. The goal is that the ANN module learns to mimic the navigation decision-making demonstrated by the expert system of section 3.5. The problem is addressed with dataset aggregation, which is a type of interactive direct policy learning (see section 2.1). In the learning process, rollouts to generate additional training data alternate with supervised trainings of the ANN module on the aggregated dataset. During a rollout, the drone navigates through a racetrack based on the navigation decisions of the ANN module. Whenever the ANN module makes a navigation decision that would cause the drone to deviate too far from the expert's global trajectory through the racetrack, the interactive expert system intervenes and provides an expert navigation decision that is executed instead. Also, a training sample labeled with the expert navigation decision is added to the training dataset so that, during training, the ANN module can learn from the mistakes made during rollout.

The user specifies the learning configuration (see table 4.3) for the individual ANN module variant, which includes the rollout and the training configuration. The rollout configuration comprises a set $\mathcal{S}$ of simulation configurations (see table 4.1), a set $\mathcal{V}$ of maximum drone speeds (see equ. 3.46) and a set $\mathcal{E}$ of pairs of margins and thresholds for the expert intervention share (hereafter referred to as margin-threshold pairs). A margin determines the distance of the drone from the global trajectory above which the expert intervenes. The threshold determines the share of expert navigation decisions in the total navigation decisions of a rollout, below which a rollout configuration is considered sufficiently learned. The training configuration comprises the sequence length of the training samples aggregated during rollout (which must be one if the individual ANN module is not recurrent), the number of epochs after each rollout, the batch size, the loss function, the optimizer type and the scheduling of the learning rate.

Table 4.3: Learning configuration options

Learning	Rollout	Simulation configurations $\mathcal{S}$
		Max. drone speeds $\mathcal{V}$
		Margin-threshold pairs $\mathcal{E}$
	Training	Sequence length $\breve{N}^{\text{seq}}$
		Number of Epochs $\breve{N}^{\text{epoch}}$
		Batch size $\breve{N}^{\text{batch}}$
		Loss
		Optimizer
		Learning rate

In detail, the learning process proceeds in the following steps.

- For every margin-threshold pair (M, T) in $\mathcal{E}$
- For every simulation configuration S in $\mathcal{S}$
- For every maximum drone speed V in $\mathcal{V}$

1. Process and load S in the simulation.
2. Set the maximum drone speed $\breve{v}^{\mathrm{d}}_{\mathrm{max}} = v$ of the planning module.
3. Compute the expert's global trajectory through the racetrack.
4. Roll out the ANN module with the interactive expert system for one round on the racetrack. At the user-specified main frequency $\breve{f}^{\mathrm{main}}$, the following steps are taken.
   (a) The latest data from the onboard sensors is preprocessed to a single (non-sequential) input. (Which sensor data is included depends on the configuration of the individual ANN module.)
   (b) The ANN module processes the single input to make a navigation decision. (If the individual ANN module is recurrent, it can still make temporal connections because the single inputs incoming at the frequency $\breve{f}^{\mathrm{main}}$ constitute a time series.)
   (c) The planning module computes the local trajectory for the ANN navigation decision.
   (d) If the end position of the local trajectory is more distant from the expert's global trajectory than M:
      i. The expert system intervenes by making a navigation decision based on its knowledge.
      ii. The planning module re-computes the local trajectory for the expert navigation decision.
   (e) The local trajectory is forwarded to the control stack, which tracks it at a higher frequency than $\breve{f}^{\mathrm{main}}$.
5. For every expert intervention of the latest rollout, add a sample to the training dataset. A sample comprises an expert navigation decision as a label and the corresponding sequence of inputs for the individual ANN module variant. The sequence starts $\breve{N}^{\mathrm{seq}}$ time steps of duration $1/\breve{f}^{\mathrm{main}}$ back in time and ends at the time step where the expert made the navigation decision. Record the share of the expert navigation decisions in the total (expert and ANN) navigation decisions made during the rollout.

6. Train the ANN module on the aggregated training dataset with supervised learning. The number of epochs $\breve{N}^{\text{epoch}}$, the batch size $\breve{N}^{\text{batch}}$, the loss function, the optimizer and the learning rate scheduling are specified in the training configuration. (For a recurrent ANN module, the samples of the training dataset are usually sequential. The ANN module then operates in many-to-one mode, whereby only the navigation decision from the processing of the last input of the input sequence is used to calculate the loss.)

7. If the recorded expert intervention share (from step 5) is greater than T, go back to step 1. Else the current (M, T)-S-V combination in the rollout configuration is considered as sufficiently learned by the ANN module.

4.3 Race Tests

After an ANN module variant completed the imitation learning process, its race performance is tested. To do this, the ANN module variant is rolled out with the expert system deactivated. From records made during the rollout, the variant's race performance is evaluated.

The user specifies the testing configuration (see table 4.4), which includes a set of simulation configurations, a set of maximum drone speeds, and the number $\breve{N}^{\text{rep}}$ of rollout repetitions for a given combination of simulation configuration and maximum drone speed. In order to ensure comparability of the race performance of different ANN module variants while also using randomized racetracks, $\breve{N}^{\text{rep}}$ randomized racetracks are pre-computed for every possible simulation configuration (see table 4.1).

Table 4.4: Testing configuration options

Testing	Simulation configurations $\mathcal{S}$
	Max. drone speeds $\mathcal{V}$
	number of repetitions $\breve{N}^{\text{rep}}$

In detail, the race tests are conducted as follows.

- For every simulation configuration S in $\mathcal{S}$
- For every maximum drone speed V in $\mathcal{V}$
- For every repetition N in $\mathcal{N}$

1. Load S with the N-th pre-computed racetrack for S in the simulation.
2. Set the maximum drone speed $\breve{v}^{\text{d}}_{\text{max}} = v$ of the planning module.

3. Roll out the ANN module for one round on the racetrack. At the user-specified main frequency $\breve{f}^{\text{main}}$, the following steps are taken.

   (a) Record the time-stamped position of the drone.

   (b) The latest data from the onboard sensors is preprocessed to a single (non-sequential) input. (Which sensor data is included depends on the configuration of the individual ANN module.)

   (c) The ANN module processes the single input to make a navigation decision. (If the individual ANN module is recurrent, it can still make temporal connections because the single inputs incoming at the frequency $\breve{f}^{\text{main}}$ constitute a time series.)

   (d) The planning module computes the local trajectory for the ANN navigation decision.

   (e) The local trajectory is forwarded to the control stack, which tracks it at a higher frequency than $\breve{f}^{\text{main}}$.

4. Record if the drone, during the latest rollout, completed the racetrack by traversing all gates without crashing.

The recordings allow the race performance of an ANN module variant to be evaluated. On the basis of the racetrack completion records, a variant's racetrack completion share on a set of simulation configurations depending on the maximum drone speed can be calculated. The racetrack completion share quantifies the robustness of a variant's navigation decision-making for a given setup. The time-stamped drone position records of the rollouts reproduce the flight trajectories induced by a variant. The optimality with respect to jerk and snap of these trajectories and therewith the variant's navigation decision-making can be quantified with the loss functions of the optimization problem formulations of the global and the local trajectory (see equ. 3.61 and 3.52).

4.4 Design of Experiment 1

Experiment 1 studies the race performance of two feedforward and three recurrent ANN module variants on the randomized figure-8 racetrack in a single simulation environment. Table 4.5 shows the configuration of experiment 1, which includes the ANN module configuration and the rollout and training configuration of the imitation learning process for all variants examined. A variant's ANN module configuration determines its number of trainable parameters and multiply-accumulate (MAC) operations at a single inference. Table 4.6 shows these numbers obtained with torchinfo[10] for all variants

[10] `https://github.com/TylerYep/torchinfo`, accessed on October 5, 2022

examined. A variant's number of trainable parameters determines its degree of freedom to fit the aggregate training data and, thus, can correlate with how well the variant performs at training and eventually at race. For this reason, the numbers of trainable parameters are taken into account when comparing the performance of the variants examined. Furthermore, it has a great impact on how memory- and time-consuming the variant's training is. A variant's number of MAC operations measures the computing effort of a single inference and, thus, has a great impact on the inference time on a computing platform. As drones are limited in computing power, a variant's inference time is critical at race when navigation decisions must be made at a relatively high frequency. However, this experiment can only list the MAC numbers of the variants examined without further investigations on the inference time, because the experiments in this thesis could only be conducted in simulation on a desktop computer.

Common to all ANN module variants is that first, the CNN submodule inputs 240x160 preprocessed RGB images from the drone's onboard camera. Second, the HEAD submodule is a ReLU-activated, fully-connected layer that outputs navigation decisions. And third, the input sequences of the training samples are processed with a dropout with a resultant dropout probability of 50 %. For a variant that during inference applies dropout x times with the same dropout probability and that trains on samples with the input sequence length y, the dropout probability at a single application is calculated with

$$p = 1 - \sqrt[xy]{0.5}. \tag{4.1}$$

The two feedforward variants are characterized by deactivated GRU submodule and an activated FC submodule, which consists of three ReLU-activated, dropout-subjected, fully-connected layers with a width of 256 neurons. For a resultant dropout probability of 50% when processing input sequences of length 1, equation 4.1 yields the dropout probability at a single application of approximately 20.63%. The first feedforward variant (F1) is a slightly extended version of the ANN deployed in the baseline autonomous navigation method of Kaufmann et al. [31]. The CNN submodule of F1 is, like the baseline, implemented with an 8-layer Resnet [20]. Unlike the baseline, its FC submodule has three instead of one layer. This extension adjusts F1 to the other examined variants in terms of the number of trainable parameters of the FC/GRU submodule in order to increase the variants' comparability. The second feedforward variant (F2) differs from F1 only in the CNN submodule as it uses a 14-layer instead of an 8-layer Resnet. Preliminary experiments on Resnet implementations of different complexity (not documented) suggest that more complex Resnets than the one used in the baseline work yield significantly better results. The Resnet14 was chosen for F2 because it represents a good compromise in terms of the increase in trainable parameters, thus keeping the variant's memory occupation and training duration within tolerable limits. Nonetheless, using the 8-layer or the 14-layer Resnet has by far the largest impact on the total number

of both parameters and MAC operations of a variant. Compared to Resnet8, Resnet14 has approximately 9 times more parameters and performs approximately 20 times more MAC operations at a single inference. F1 is the only variant using Resnet8 and has thus by far the lowest MAC number. This experiment compares F1 and F2 to investigate whether the higher CNN complexity of F2 is reflected in the race performance of the variant.

The three recurrent variants are characterized by a deactivated FC submodule and an activated GRU submodule, which consists of three layers with a hidden size of 64, of which the second and the third layer are subjected to dropout. For a resultant dropout probability of 50% when processing input sequences of length 25, equation 4.1 yields the dropout probability at a single application of approximately 1.38%. Care was taken to ensure that the GRU submodule of the three recurrent variants has fewer trainable parameters than the FC submodule of the two feedforward variants, in order to rule out the possibility that the recurrent variants perform better only because of a higher number of trainable parameters. All three variants use the Resnet14 because F2 performs significantly stronger than F1 (see results in section 5.1). The first recurrent variant (R1) is the recurrent equivalent of F2. The comparison of F2 and R1, thus, aims to investigate the impact of the GRU submodule's temporal comprehension on the race performance of a variant. The second recurrent variant (R2) differs from R1 with respect to the CAT submodule. While the CAT submodule for R1 is deactivated, for R2, it inputs all available optional inputs. The comparison of R1 and R2, thus, aims to investigate the impact of using the optional inputs within a variant's navigation decision-making on the variant's race performance. The third recurrent variant (R3) differs from R1 with respect to the CNN submodule. While the CNN submodule for R1 is trainable and not pretrained, for R2, it is pretrained and only partly trainable. The pretraining of the CNN submodule was carried out with a preliminary final layer on training data from preliminary experiments (not documented). The CNN is only trainable at the single trainings whenever a margin-threshold pair is completed in the learning process. As a result, R3 has by far the lowest number of trainable parameters for most trainings, whereby the learning of R3 can be highly accelerated. The comparison of R1 and R2, thus, aims to investigate whether this shortcut has a tolerable impact on a variant's race performance.

To summarize the relation of the ANN module configurations of the variants examined: F1 represents the feedforward ANN of the baseline work. F2 integrates a 14-layer instead of an 8-layer Resnet. R1 is the recurrent counterpart of F2. R2 additionally uses the optional inputs. The CNN submodule of R3 is pretrained and trainable only partly in time.

The rollout configuration of the learning process is the same for all variants examined. The variants learn to navigate through the randomized, counterclockwise figure-8 racetrack built with the TUB-DAI or THU-DME gate type. The racetrack is thereby located in site A of the spaceship interior scene. This limitation to a single simulation

environment is motivated by the high time expenditure of the imitation learning process. As a result, this experiment can only compare the variants in terms of their ability to generalize to the randomized figure-8 racetrack located in a single, fixed simulation environment and does not provide insights regarding the generalization to simulation environments unseen in the learning process. The maximum drone speeds of the planning module during the learning rollouts range from 4 to 10 m/s in 1 m/s steps. The learning at different speeds is motivated by the fact that the maximum drone speed influences the state distribution of a variant's rollout. For the intervening expert system, there are three margin-threshold pairs, with the margins becoming wider and the thresholds becoming more stringent. To complete the learning on a rollout configuration, the variant must perform a rollout with less than 10/5/1% expert interventions that are triggered whenever the variant would navigate the drone further than 0.5/0.75/1.0 m from the expert's global trajectory.

With respect to the training configuration of the learning process, all variants share that they determine loss with the standard SmoothL1Loss[11] PyTorch implementation and update the variants' trainable parameters accordingly with the standard ADAM[12] PyTorch implementation, whereby the learning rate is exponentially scheduled with the initial value of 10^{-4} for each training and a decay of 95% per epoch. Moreover, all variants share that the batch size is set to the maximum value containable by the GPU memory of my desktop computer. The feedforward and the recurrent variants differ by the aggregate training samples and the number of epochs per training. While for the feedforward variants, the inputs of the samples are non-sequential, for the recurrent variants, they have a sequence length of 25. As a result, the training epochs of the recurrent variants consume significantly more time. However, preliminary experiments (not documented here) suggested that these recurrent training epochs also lower the loss more effectively. Therefore, the number of epochs per training is set to 10 and 3 for the feedforward and the recurrent variants, respectively.

After the learning process, the variants are tested. Table 4.7 shows the configuration of the race test. For testing, the variants roll out on the same simulation configuration (environment and racetrack) with the same maximum drone speeds as for learning. For every combination in the testing configuration, each variant rolls out 10 times.

4.5 Design of Experiment 2

Experiment 2 takes the ANN module variant R1 of experiment 1 (see table 4.5) as a starting point to study the impact of the depth of the GRU submodule on a variant's race performance. The starting point R1 exhibits the best race performance among all

[11] `https://pytorch.org/docs/stable/nn.html`, accessed on October 5, 2022

[12] `https://pytorch.org/docs/stable/optim.html`, accessed on October 5, 2022

variants examined in experiment 1 (see section 5.1). The following briefly recalls the configuration of R1 in experiment 1.

R1 integrates a trainable, not pretrained 14-layer Resnet, a three-layer GRU with a hidden size of 64 and a resultant dropout probability of 50% and a final, ReLU-activated, fully-connected layer in order to map 240x160 preprocessed RGB images from the drone's onboard camera to navigation decisions forwarded to the planning module. The variant rolls out on the randomized figure-8 racetrack in a single simulation environment for maximum drone speeds of $4, 5, \ldots, 10$ m/s. Thereby, a training sample of sequence length 25 is generated whenever the expert system intervenes to correct a navigation decision that would navigate the drone out of the current margin. After each rollout, R1 trains on the aggregate training dataset for 3 epochs with supervised learning.

Experiment 2 studies the race performance of five ANN module variants that are configured like R1 (see table 4.5) except that the GRU submodule has 1, 2, 3 (this is the original number of R2), 5, or 10 layers and the dropout at a single application has a probability of approximately none (by design, the first layer applies no dropout), 2.73, 1.38, 0.69 or 0.31% , respectively, in order to maintain a resultant dropout probability of 50% (see equ. 4.1). A variant examined in this experiment with L GRU layers of a hidden size of H is referred to as R1-LxH.

The five variants are trained for 200 epochs on the final training dataset of R1 which contains 18k samples collected in 114 rollouts of the learning process. Thereafter, the variants perform race tests with the same configuration as R1 (see table 4.7).

4.6 Design of Experiment 3

Experiment 3 studies the race performance of a feedforward and a recurrent ANN module variant on the randomized gap racetrack in several simulation environments. Table 4.8 shows the configuration of experiment 3, including the ANN module configuration and the learning configuration. Table 4.9 shows the number of trainable parameters and MAC operations for both examined variants.

Both variants integrate a trainable, not pretrained 18-layer Resnet to process 360x240 RGB images, the CAT submodule to input all available, optional inputs and the ReLU activated HEAD submodule to output navigation decisions. Moreover, both variants process a training sample input with a resultant dropout probability of 50% (see equ. 4.1).

The feedforward variant (E3F) is characterized by the deactivated GRU submodule and the activated FC submodule, which is a single, ReLU activated, dropout-subjected, fully-connected layer with 512 neurons. Therewith, E3F corresponds to the ANN design in the baseline work with the Resnet8 extended to a Resnet18. The recurrent variant (E3R) is characterized by the deactivated FC submodule and the activated GRU sub-

module, which consists of three layers with a hidden size of 16, of which the second and the third layer are subjected to dropout.

The Resnet18 dominates the numbers of trainable parameters and (even more) MAC operations for both variants. Nonetheless, care was taken that the GRU submodule of E3R has less trainable parameters than the FC submodule of E3F in order to rule out the possibility that E3R performs better only because of a higher complexity.

The rollout configuration of the learning process is the same for both variants examined. For a maximum drone speed of 4, 6 and 8 m/s, the variants learn to navigate through the randomized, counterclockwise and clockwise gap racetrack built with the TUB-DAI or THU-DME gate type, which is located in all three sites of the scenes spaceship interior, destroyed city, industrial site and polygon city. For the intervening expert system, there are two margin-threshold pairs. The first margin is wider than the second, while the threshold remains to be 6%.

The training configuration of the learning process is partly the same for both variants. Both are trained for 5 epochs with the SmoothL1Loss, the ADAM optimizer and an exponentially scheduled learning rate. The batch size for both variants is set to the maximum value containable by the GPU memory of my desktop computer. E3F trains on samples with non-sequential input, while E3R trains on samples with an input sequence length of 3.

Table 4.10 shows the configuration for the race tests. The variants roll out in the four scenes seen during learning as well as the desert mountain scene unseen during learning. Thereby, the maximum drone speed is set to 4, 5, ..., 10 m/s, of which only 4, 6 and 8 m/s were experienced during learning. For every combination in the testing configuration, each variant rolls out only once due to the large number of combinations.

4.7 Design of Experiment 4

Experiment 4 takes the better performing of both ANN module variant examined in experiment 3, i.e., the recurrent variant E3R (see table 4.8), and its final, aggregated training dataset as a starting point for studying the impact of the input sequence length and the image size of the training data on the race performance of a recurrent variant.

The training dataset of E3R obtained in experiment 3 contains 40k samples whose input is a sequence of three pairs of a 360x240 RGB image and an optional input vector. In experiment 4, the training dataset is rebuilt from the raw data recorded during the learning process of E3R with a varying input sequence length of 2, 3, 5 and 10 and a varying RGB image size of 240x160 and 360x240, which yields eight different datasets. Then, the eight variants named E3R-2*240x160, E3R-3*240x160, E3R-5*240x160, E3R-10*240x160, E3R-2*360x240, E3R-3*360x240 (the starting point), E3R-5*360x240 and E3R-10*360x240 train on these datasets. The ANN modules of these eight variants are configured like the starting point. However, for a resultant

dropout probability of 50%, the dropout probability for a single dropout application of the GRU submodule is adjusted by equation 4.1.

Finally, the variants perform race tests with the same configuration as E3R (see table 4.10).

Table 4.5: Configuration of Experiment 1

				Feedforward		Recurrent		
				F1	F2	R1	R2	R3
ANN		CNN	Input	240x160 RGB				
			Model	Resnet8	Resnet14			
			Pretrained	No				Yes
			Trainable	Yes				Partly
		CAT	Opt. Input	None			All	None
		GRU	# Layers	None		3		
			Hidden size			64		
			Dropout			0.013767		
		FC	# Layers	3		None		
			Width	256				
			Dropout	0.206299				
			Activation	ReLU				
		HEAD	Activation	ReLU				
			Output	Navigation decision				
Learning	Rollout	Environments	Scenes	Spaceship interior				
			Sites	A				
		Racetracks	Types	Figure-8				
			Generations	Randomized				
			Directions	Counterclockwise				
			Gates	TUB-DAI, THU-DME				
		Max. drone speeds		4, 5, 6, 7, 8, 9, 10 m/s				
		Margin-threshold		(0.5, 10), (0.75, 5), (1.0 m, 1%)				
	Training	Sequence length		1		25		
		# Epochs		10		3		
		Batch size		256	32	8		
		Loss		SmoothL1Loss				
		Optimizer		ADAM				
		Learning rate		Exponential: $10^{-4} \cdot 0.95^{\text{epoch}}$				

Table 4.6: Numbers (in the format $men = m \times 10^n$) of trainable parameters (TP) and multiply-accumulate operations (MAC) of the ANN module variants of experiment 1. For R3, the table does not reflect the negligible number of single trainings in the learning process where the CNN parameters are momentarily trainable.

ANN	#	F1	F2	R1	R2	R3
CNN	TP	309e3	2.78e6	2.78e6	2.78e6	0
	MAC	52.9e6	1.07e9	1.07e9	1.07e9	1.07e9
GRU	TP	0	0	112e3	113e3	112e3
	MAC	0	0	112e3	112e3	112e3
FC	TP	164e3	197e3	0	0	0
	MAC	164e3	197e3	0	0	0
HEAD	TP	771	771	195	195	195
	MAC	771	771	195	195	195
Total	TP	474e3	2.98e6	2.89e6	2.90e6	112e3
	MAC	53.1e6	1.07e9	1.07e9	1.07e9	1.07e9

Table 4.7: Testing configuration for experiment 1

Testing	Environments	Scenes	Spaceship interior
		Sites	A
	Racetracks	Types	Figure-8
		Generations	Randomized
		Directions	Counterclockwise
		Gates	TUB-DAI, THU-DME
	Max. drone speeds		4, 5, 6, 7, 8, 9, 10 m/s
	Number of repetitions		10

Table 4.8: Configuration of Experiment 3

				Feedforward	Recurrent
				E3F	E3R
ANN		CNN	Input	360x240 RGB	
			Model	Resnet18	
			Pretrained	No	
			Trainable	Yes	
		CAT	Opt. Input	All	
		GRU	# Layers	None	3
			Hidden size		16
			Dropout		0.109101
		FC	# Layers	1	None
			Width	512	
			Dropout	0.5	
			Activation	ReLU	
		HEAD	Activation	ReLU	
			Output	Navigation decision	
Learning	Rollout	Environments	Scenes	Spaceship interior, destroyed city, industrial site, polygon city	
			Sites	A, B, C	
		Racetracks	Types	Gap	
			Generations	Randomized	
			Directions	Counterclockwise, Clockwise	
			Gates	TUB-DAI, THU-DME	
		Max. drone speeds		4, 6, 8 m/s	
		Margin-threshold		(0.7, 6), (0.5 m, 6%)	
	Training	Sequence length		1	3
		# Epochs		5	
		Batch size		32	16
		Loss		SmoothL1Loss	
		Optimizer		ADAM	
		Learning rate		Exponential: $10^{-4} \cdot 0.99^{\text{epoch}}$	

Table 4.9: Numbers (in the format $me n = m \times 10^n$) of trainable parameters (TP) and multiply-accumulate operations (MAC) of the ANN module variants of experiment 3.

ANN	#	E3F	E3R
CNN	TP	11.2e6	
	MAC	3.24e9	
GRU	TP	None	29.1e3
	MAC		29.1e3
FC	TP	267e3	None
	MAC	267e3	
HEAD	TP	1.54e3	48
	MAC	1.54e3	48
Total	TP	11.4e6	11.2e6
	MAC	3.24e9	

Table 4.10: Testing configuration for experiment 3

Testing	Environments	Scenes	Spaceship interior, destroyed city, industrial site, polygon city, desert mountain
		Sites	A, B, C
	Racetracks	Types	Gap
		Generations	Randomized
		Directions	Counterclockwise, Clockwise
		Gates	TUB-DAI, THU-DME
	Max. drone speeds		4, 5, 6, 7, 8, 9, 10 m/s
	Number of repetitions		1

Chapter 5

Evaluation

This chapter comprises two sections: the first presents the results from the experiments of chapter 4, the second discusses and interprets these results.

5.1 Results

This section presents the results from the four experiments conducted. This includes, for experiment 1 and 3, the data aggregation statistics of the imitation learning process and, for all experiments, the loss trends through the imitation/supervised learning process and the racetrack completion shares over the maximum drone speeds from the race tests.

Table 5.1 shows the data aggregation statistics of the imitation learning process for the variants of experiment 1. While all three recurrent variants (R1, R2 and R3) com-

Table 5.1: Data aggregation of experiment 1

	F1	F2	R1	R2	R3
Learning completed	No	No	Yes	Yes	Yes
# Rollouts	297	208	114	136	178
# Training samples	75,939	28,814	18,083	20,823	28,005
# Validation samples	1,552	569	418	408	598

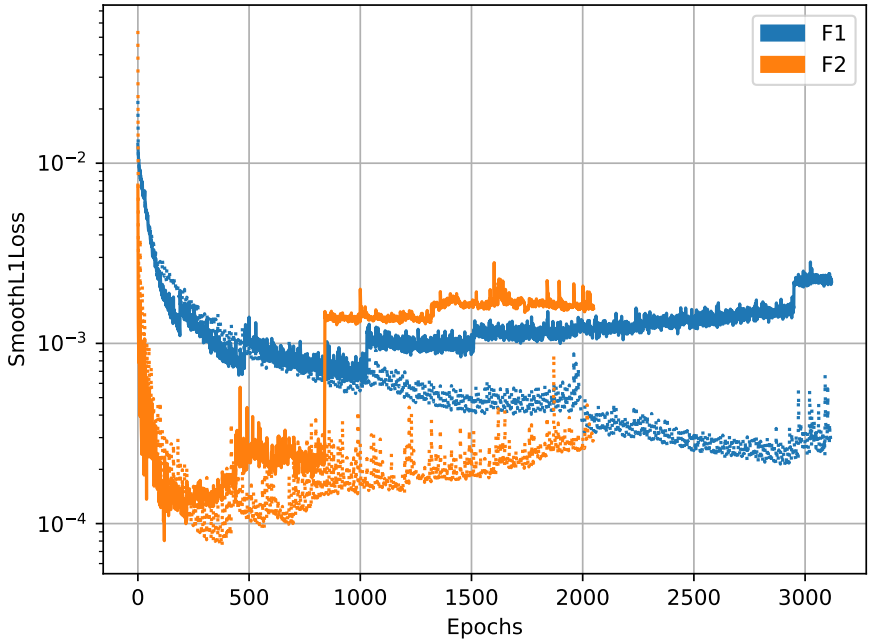
pleted the imitation learning process, both feedforward variants (F1 and F2) stopped making progress and failed this task. The learning process of F1 (or F2) was terminated at the 47th (72th) repetition of the rollout combination with a maximum drone speed of 5 m/s (10 m/s) and the last margin-threshold pair of (1.0 m, 1%). Consequently, F1 completed about 2/3 of the learning rollout combinations, whereas F2 completed all but the last combination. In the incomplete learning process, both feedforward variants rolled out more often and collected more data than all three recurrent variants in the complete learning process. F1 performed the most rollouts with 297 and aggregated the most

data with 75,939 training and 1,552 validation samples. F2 follows by a wide mar-gin with 208 rollouts and 28,814 training and 569 validation samples. In third place is F3 with 178 rollouts and only slightly fewer data with 28,005 training and 598 validation samples. R1 (and R2) performed by far the least rollouts with 408 (418) and aggregated by far the least data with 18,083 (20,823) training and 418 (408) validation samples.

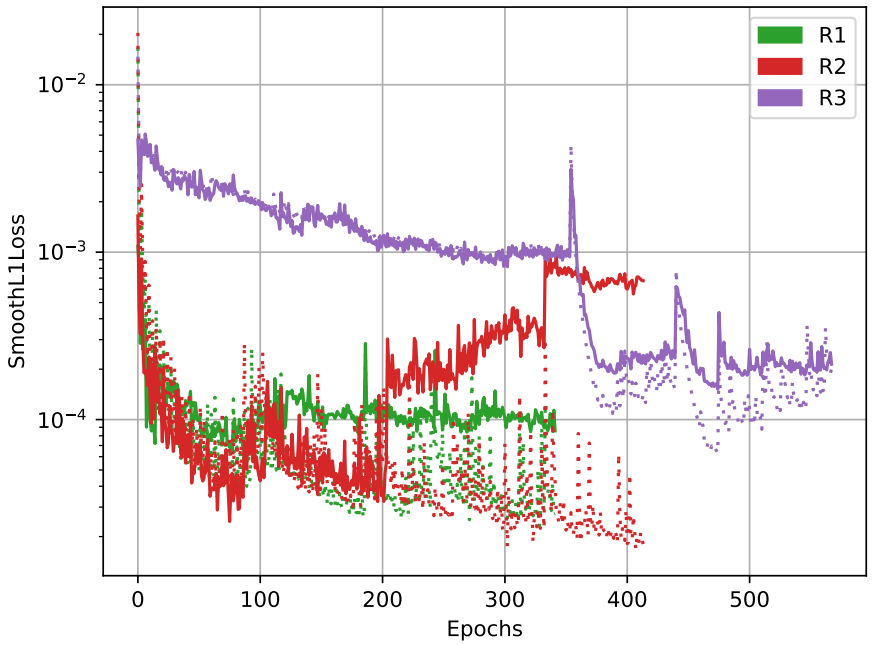
Figure 5.1 shows the training and the validation losses over the epochs of the imitation learning process for the variants of experiment 1. The training or the validation loss of a variant is calculated with the SmoothL1Loss[1] function on the training or validation dataset aggregated by that variant. The figure displays the feedforward and the recurrent variants separately, because the feedforward variants performed much more epochs than the recurrent variants in the learning process. While F1 and F2 went through about 3000 and 2000 epochs, respectively, R1, R2 and R3 required about 350, 400 and 550 epochs, respectively. At the end of the learning process, the feedforward variants achieve roughly the same losses on their aggregated data, which are higher than the final losses of the recurrent variants. F1 has a final training and validation loss of approximately 2.6×10^{-4} and 2.2×10^{-3}. F2 has a slightly higher, final training and a slightly lower, final validation loss of approximately 2.9×10^{-4} and 1.6×10^{-3}. R1 has the second lowest, final training and the lowest, final validation loss of approximately 2.7×10^{-5} and 8.6×10^{-5}. R2 has the lowest final training loss and the highest validation loss among the recurrent variants of approximately 1.8×10^{-5} and 6.7×10^{-4}. R3 has a final training and validation loss of approximately 2.0×10^{-4} and 2.1×10^{-4}, which are the closest final losses to each other. From the lowest to highest, the approximate ratios of validation to training losses are 1.1 for R3, 3.2 for R1, 5.5 for F2, 8.5 for F1 and 37.2 for R2.

Figure 5.2 shows the racetrack completion shares (RCS) over the maximum drone speeds of the race tests for the variants of experiment 1. The RCS of a variant is the share of the rollouts where the variant completed the racetrack in all rollouts conducted with the same maximum drone speed in the race test of that variant. For all variants, there is a tendency for the RCS to decrease with higher maximum speeds. This culminates in the fact that no variant completed any racetrack for the highest maximum drone speed tested of 10 m/s. At the maximum drone speed of 9 m/s, all three recurrent variants perform better than the feedforward variants. Over all maximum speeds, F1 has by far the lowest RCS, which is roughly 50% for lower speeds from 4 to 6 m/s and 0% for faster speeds from 7 to 10 m/s. F2, R2 and R3 have roughly the same RCS over the tested maximum speeds, which ranges from 70 to 85% for maximum speeds from 4 to 7 m/s and for higher maximum speeds decreases almost linearly to 0% for 10 m/s. The RCS of R1 is about the same as F2, R2 and R3 for the maximum drone speed of 4 m/s. For maximum speeds from 5 to 9 m/s, however, it is approximately 20 percentage points

[1] `https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html`, accessed on October 5, 2022



(a) Feedforward variants



(b) Recurrent variants

Figure 5.1: Training (dotted) and validation (solid) losses of experiment 1

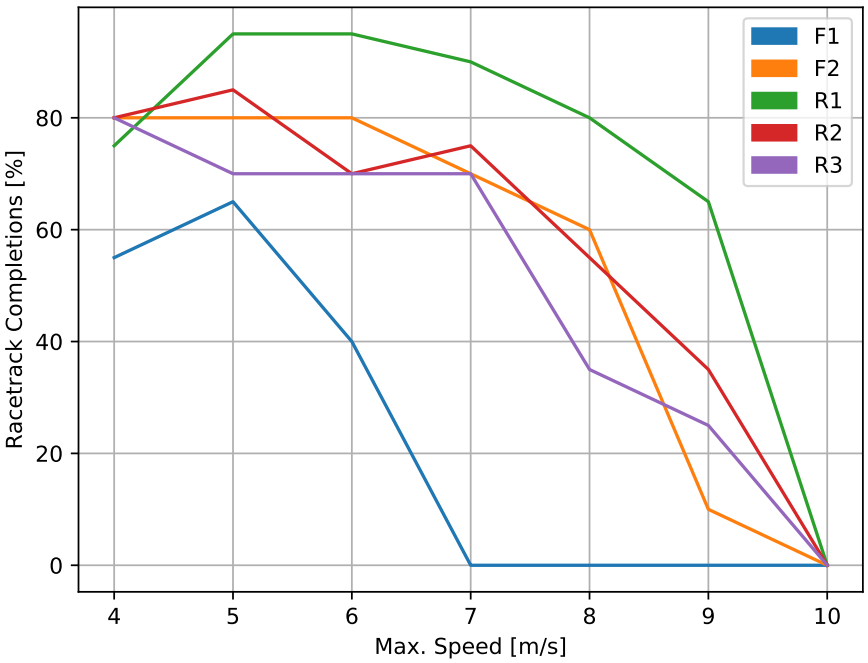


Figure 5.2: Racetrack completion shares of experiment 1

higher.

Figure 5.3 shows the training and the validation losses over the epochs of the supervised learning process for the variants of experiment 2. The training or the validation loss of a variant is calculated with the SmoothL1Loss1 function on the final training or validation dataset of R1 in experiment 1. All variants train for 200 epochs. The single-layer GRU variant (R1-1x64) achieves the lowest final training and the highest final validation loss of approximately 5.8×10^{-6} and 1.4×10^{-4}. The 10-layer GRU variant (R1-10x64), which has the most GRU layers in experiment 2, achieves the highest final training and the lowest final validation loss of approximately 2.3×10^{-5} and 6.5×10^{-5}. The variants in between (R1-2x64, R1-3x64 and R1-5x64) achieve roughly the same final training and the same final validation loss of approximately 1.3×10^{-5} and 1.0×10^{-4}. Consequently, from lowest to highest, the approximate ratios of validation to training losses are 2.8 for R1-10x64, 7.7 for R1-2x64, R1-3x64 and R1-5x64 and 24.1 for R1-1x64. The validation loss of R1-10x64 converges with a higher fluctuation than the validation losses of the other variants.

Figure 5.4 shows the RCS over the maximum drone speeds for the variants of experiment 2. R1-1x64, R1-2x64, R1-3x64 and R1-5x64 have roughly the same RCS, which ranges from 85 to 100% for maximum speeds from 4 to 8 m/s, from 65 to 85% for 9 m/s and from 5 to 25% for 10 m/s. For maximum speeds from 9 to 10 m/s, however, the RCS of R1-3x64 and R1-5x64 is approximately 10 percentage points higher than

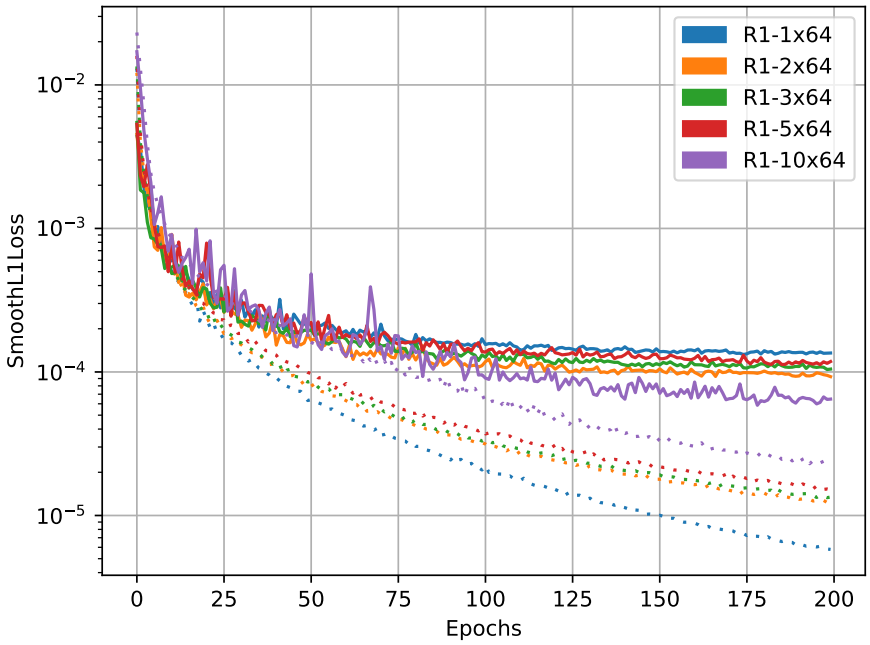


Figure 5.3: Training (dotted) and validation (solid) losses of experiment 2

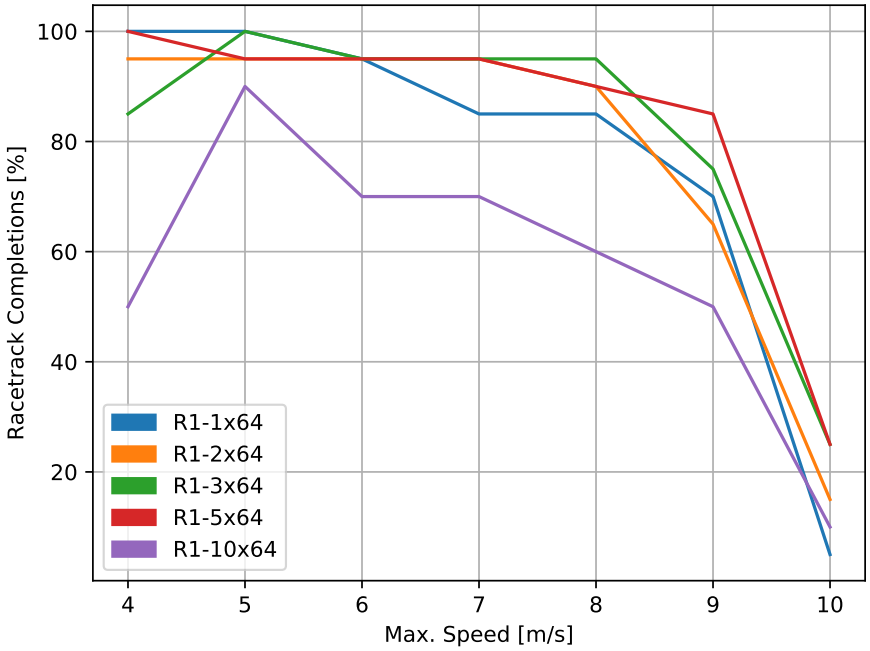


Figure 5.4: Racetrack completion shares of experiment 2

R1-1x64, R1-2x64. Except for the maximum speed of 10 m/s, R1-10x64 has the by far the lowest RCS, which is about 20 percentage points lower over the tested maximum speeds.

Table 5.2 shows the data aggregation statistics of the imitation learning process for the variants of experiment 3. Both variants, the feedforward and the recurrent variant,

Table 5.2: Data aggregation of experiment 3

	E3F	E3R
Learning completed	Yes	Yes
# Rollouts	547	840
# Training samples	25,470	40,216
# Validation samples	None	None

completed the imitation learning process. Neither variant aggregated validation data because this feature had not yet been implemented when experiment 3 was conducted. The feedforward variant (E3F) performed fewer rollouts with 547 and aggregated fewer data with 25,470 training samples than the recurrent variant (E3R) which rolled out 840 times and aggregated 40,216 training samples.

Figure 5.5 shows the training losses over the epochs of the imitation learning process for the variants of experiment 3. The training loss of a variant is calculated with the SmoothL1Loss function on the training dataset aggregated by that variant. E3F trained for fewer epochs with approximately 3000 and achieved a lower final training loss of 1.3×10^{-5} than E3R with approximately 4500 and 3.2×10^{-5}. In contrast to experiment 1 (see figure 5.1), the variants of experiment 3 continued to train after the imitation learning process was completed until the training loss converged. This can be recognized by the smoother curves at the end of the loss trends. E3R makes a larger loss drop there because the value of the dropout probability for a single application had to be corrected to achieve a resultant dropout probability of 50% for both variants. The training loss trend of E3F fluctuates stronger than E3R during the imitation learning process.

Figure 5.6 shows the RCS over the maximum drone speeds of the race tests for the variants of experiment 3. Thereby, the figure distinguishes between simulation environments seen and unseen in the imitation learning process. In unseen environments, both variants have a RCS of about 0% over all maximum speeds. In seen environments, E3R has a significantly higher RCS than E3F over the tested maximum speeds. The RCS of E3F decrease almost linearly from 50% for 4 m/s to 0% for 10 m/s. The RCS of E3R ranges from 63 to 73% for maximum speeds from 4 to 7 m/s and, from there, decreases almost linearly to 5% at 10 m/s.

Figure 5.7 shows the training losses over the epochs of the supervised learning process for the variants of experiment 4. The training loss of a variant is calculated with

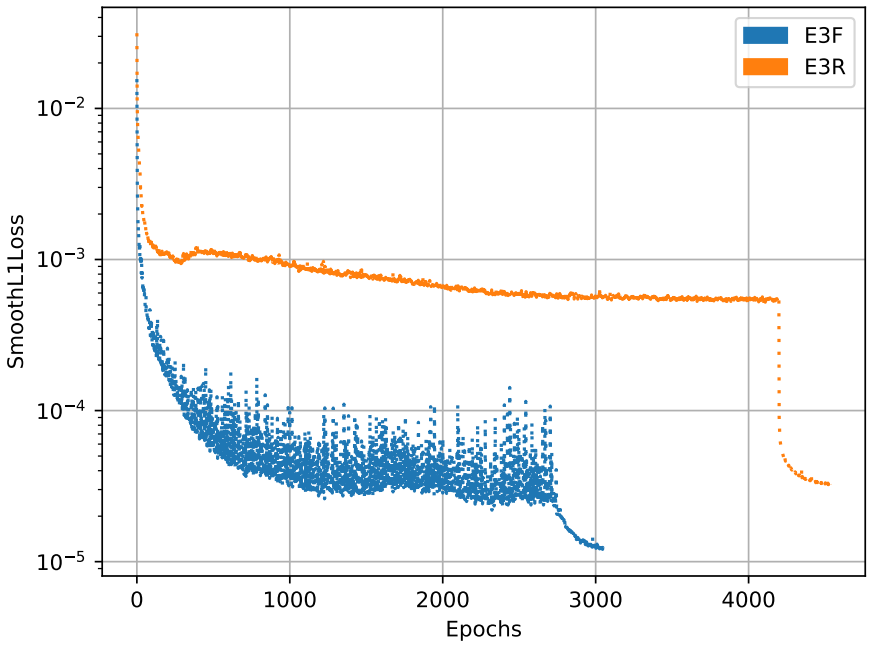


Figure 5.5: Training losses of experiment 3

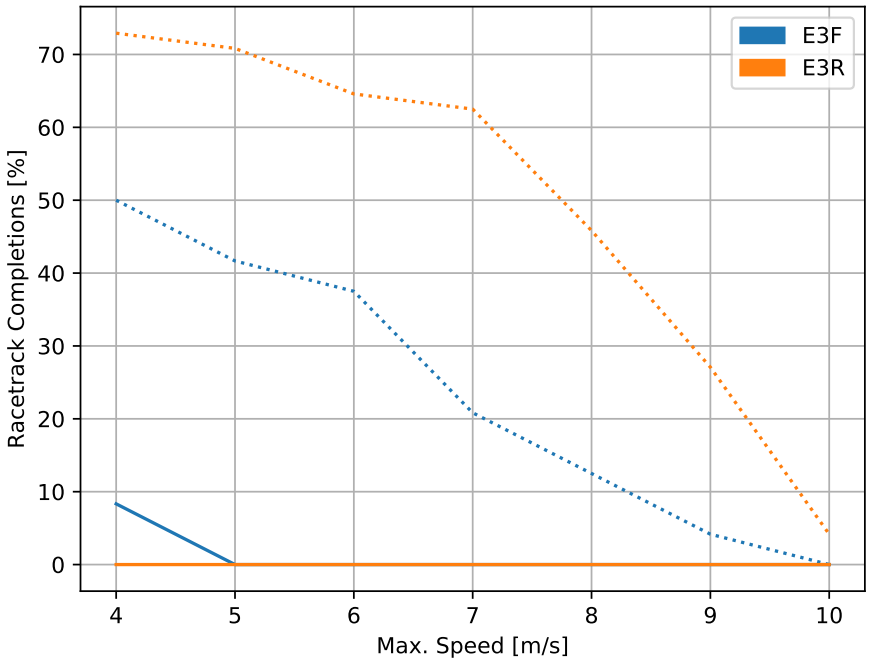
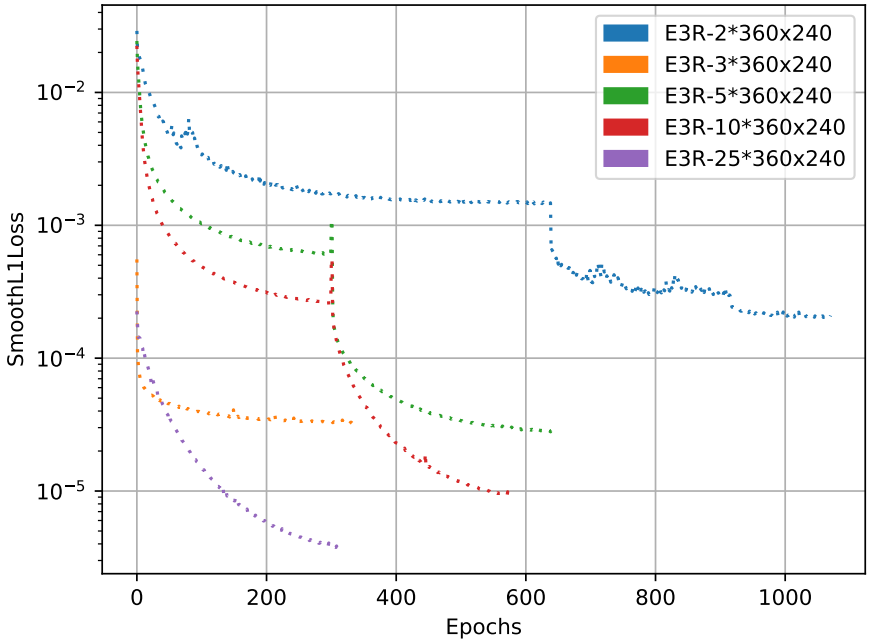
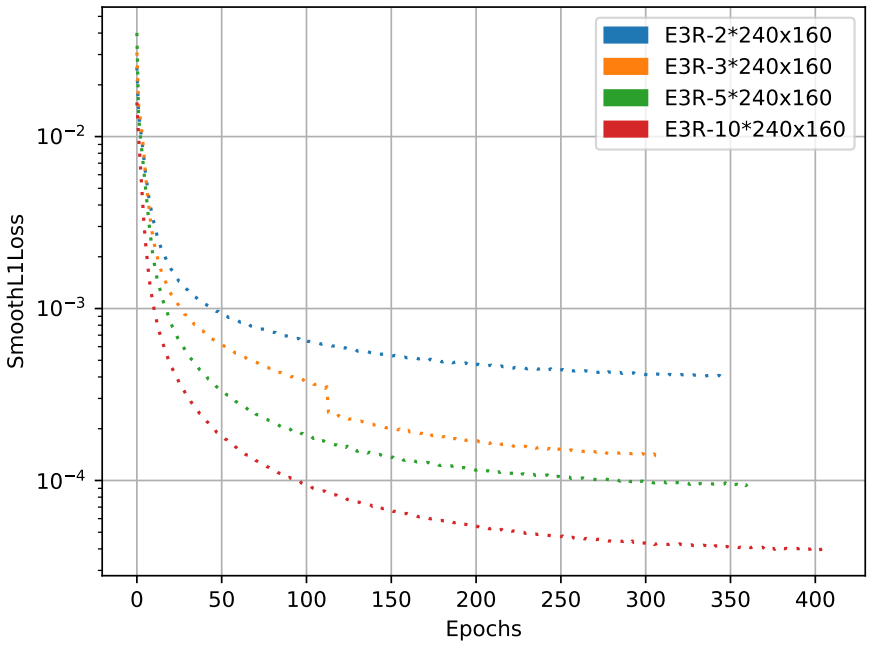


Figure 5.6: Racetrack completion shares for simulation environments seen (dotted) and unseen (solid) during learning of experiment 3



(a) 360x240 RGB images



(b) 240x160 RGB images

Figure 5.7: Training losses of experiment 4

the SmoothL1Loss function on the final training dataset of E3R in experiment 3. All variants trained close to convergence. Drops in the loss trends trace back to corrections of the dropout probability for a single application to achieve a resultant dropout probability of 50% for all variants. The variants with the RGB image size of 360x240 have lower training losses than their counterparts with the RGB image size of 240x160. For both RGB image sizes, the longer the input sequence length of the training samples, the lower the training loss. The loss trend of E3R-2*360x240 fluctuates stronger than for the other variants.

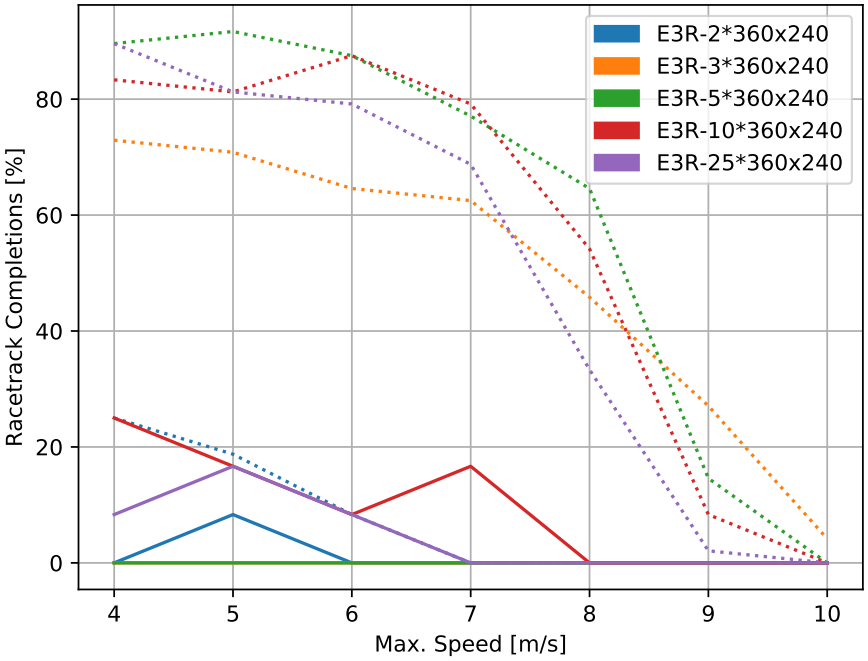
Figure 5.8 shows the RCS over the maximum drone speeds of the race tests of experiment 4. The figure distinguishes between simulation environments seen and unseen in the imitation learning process. In unseen environments, all variants have an RCS close to 0%, except for E3R-10*360x240, E3R-25*360x240, E3R-2*240x160 and E3R-10*240x160, whose RCS is up to 25, 20, 50, 35% for maximum speeds up to 7, 6, 8, 7 m/s, respectively. In seen environments, all variants have roughly the same RCS over the tested maximum speeds, which from about 80 to 90% for 4 m/s decreases almost parabolically to 0 to 15% at 9 m/s. Thereby, two variants stand out. The RCS of the starting point variant (E3R-3*360x240) is slightly lower at lower maximum speeds from 4 to 7 m/s and a slightly higher at higher speeds from 9 to 10 m/s and the RCS of E3R-2*360x240 is below 25% for all tested maximum speeds.

5.2 Discussions

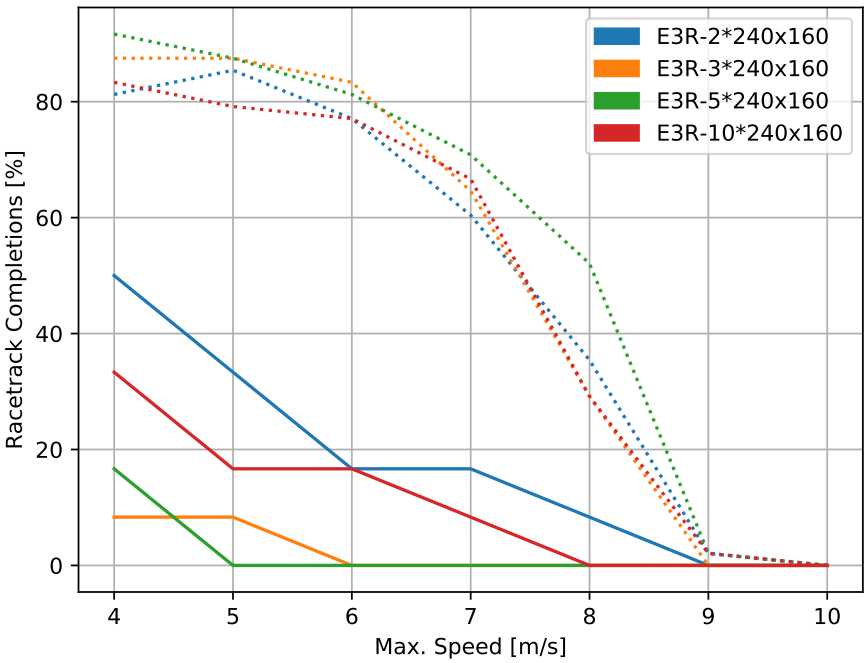
This section discusses and interprets the experimental results presented in the previous section. The performance of the ANN module variants is evaluated based on the data aggregation statistics and the loss trends from the learning process, as well as the racetrack completion shares (RCS) over maximum drone speeds from the race tests. The symbol $\sim$ denotes that a number is an approximate value.

Experiment 1 compares the variants by their learning ability for generalization to the randomized figure-8 racetrack in a fixed simulation environment. The layout of the figure-8 racetrack places three demands on navigation: the autonomous navigation method must be able to fly the drone through both, left and right, turns and across intersections. Intersections are especially difficult, because multiple gates appear in the drone's field of view, which can lead to ambiguities in the navigation decision-making.

The first feedforward variant of experiment 1 (F1), which resembles the ANN of the baseline work, could not learn these navigational qualities required by the figure-8 racetrack. As F1 stopped making progress, the imitation learning process of F1 was terminated. In the incomplete learning process, F1 still collected about three times more data than the other variants of experiment 1, with $\sim$ 76k training samples. Nonetheless, F1 performed by far the worst in the race tests of experiment 1 with a RCS of only $\sim$ 50% for maximum speeds $\leq$ 6 m/s and 0% for maximum speeds above. This



(a) RGB image size of 360x240



(b) RGB image size of 240x160

Figure 5.8: Racetrack completion shares of experiment 4

poor performance of F1 contrasts with the results from the simulated experiments of the baseline work on a static racetrack and a dynamic racetrack with sinusoidally moving gates. On the static racetrack, the baseline method aggregated $\sim 74\%$ fewer training samples than F1 but completed the racetrack at 100, ~ 85, ~ 65 and $\sim 30\%$ for maximum speeds ≤ 9 and of 10, 11 and 12 m/s, respectively. The by far worse race test performance of F1 could be attributed to the more difficult learning and testing conditions in experiment 1. While the baseline method rolls out on the same deterministic racetrack, F1 faces a slightly different randomized racetrack at each rollout in the imitation learning process and the race tests. However, considering the baseline results on the dynamic racetrack, this cannot be the only reason for the performance drop of F1. For learning to navigate through the dynamic racetrack, the baseline method rolled out on randomized racetracks like in experiment 1, whereby it collected $\sim 32\%$ more training samples than F1. At the race tests, the baseline method completed the dynamic racetrack with not too large gate movement amplitudes at 100% for a maximum speed of 8 m/s. In contrast, F1 did not complete a single racetrack for maximum speeds ≤ 7 m/s, even though it performed the race tests on only static racetracks. The fact that F1 collected fewer data than the baseline method cannot explain the performance difference because first, F1 already stalled in the imitation learning process and second, the other variants of experiment 1 performed significantly better than F1 even though they collected by far fewer data. The main reason for the performance difference is likely the different layouts of the racetrack. The racetrack of the baseline work consists of 8 gates arranged in a circle, which requires the baseline method of only learning to fly the drone through either left or right turns. In contrast, F1 must deal with turns of both directions and intersections of the figure-8 racetrack. Another reason could be that the simulation of this thesis is more photo-realistic than the simulation in the baseline work. The low complexity of the Resnet8 integrated with the baseline ANN and F1 may only infer sufficiently from the less detailed, non-photorealistic images of the baseline simulation. This argument is supported by the fact that the other variants of experiment 1, which all integrate the more complex Resnet14, perform significantly better than F1. There is another distinguishing factor in experiment 1 from the baseline experiments: F1 inputs RGB images with the size of 240x160, whereas the baseline ANN of 300x200. However, the results of experiment 4, which examined variants of two image sizes, suggest that this slight difference in the image size has, if at all, only a marginal impact on the race test performance.

The second feedforward variant of experiment 1 (F2), which differs from F1 in integrating the more complex Resnet14 instead of the Resnet8, also stalled in the imitation learning process but performs significantly better than F1 in the race tests. F2 achieved an RCS from ~ 60 to $\sim 80\%$ for maximum speeds ≤ 8 m/s and $\sim 0\%$ for maximum speeds above. Considering that the final training and validation losses of F1 and F2 are almost equal, the question arises why F2 outperforms F1 in the race tests, espe-

cially since F2 aggregated $\sim 62\%$ fewer training samples than F1. The answer is that F1 stalled at about 2/3 of the rollout combinations of the imitation learning process, while F2 stalled only at the very last combination. Thus, the training and validation data aggregated by F2, although containing much fewer samples, represent the state distribution at the rollouts of the race tests more closely. The early stages of the learning process, where the data quantity and quality are more comparable, show that F2 can reduce the training and the validation loss more and faster than F1. Overall, the comparison of F1 and F2 finds that a more complex CNN architecture can learn more accurately (lower losses on comparable data, same losses on more comprehensive data) and effectively (fewer rollouts/epochs and data) in the imitation learning process, which results in significantly better performance in the race tests. This comparison does not consider whether a drone with limited computational power could run the more complex network at the required frequency.

The first recurrent variant of experiment 1 (R1) differs from F2 in integrating three GRU instead of three fully-connected layers. R1 completed the imitation learning process in contrast to F2, whereby R1 aggregated $\sim 38\%$ fewer training samples and $\sim 45\%$ fewer rollouts. This shows that R1 needs substantially fewer data to learn a more accurate navigation decision-making than F2. Because rollouts are costly and dangerous in the real-world, R1 is more workable in real-world imitation learning than F2. Considering that the recurrent variants trained for only three while the feedforward variant trained for ten epochs per rollout, R1 could further reduce the data and number of rollouts in the comparison with F2. However, this would make the learning process of R1 much more time-consuming compared to F2, since the samples of R1 have an input sequence of 25 RGB images, while the samples of F2 have only a single RGB image input. In $\sim 83\%$ fewer epochs, R1 achieved a $\sim 91\%$ lower final training loss and a $\sim 95\%$ lower final validation loss than F2. This results in a $\sim 42\%$ lower validation-training loss ratio of R1, while both variants have a resultant dropout probability of 50% during training. The much lower losses of R1 suggest that a navigation decision is indeed temporally related to past visual observations from the drone's onboard camera and that R1, as it is recurrent, learns to leverage these underlying temporal connections for a more accurate navigation decision-making. The lower validation-training loss ratio of R1 suggests further that incorporating these temporal connections abstracts the navigation decision-making more and thus enhances the generalization ability of R1. R1 could transfer its better learning performance to a better race test performance. The RCS of R1 is ~ 20 percentage points higher than F2 for maximum speeds from 5 to 8 m/s and ~ 55 percentage points for a maximum speed of 9 m/s. Overall, the comparison of F2 and R1 indicates that the combined spatial and temporal (instead of a mere spatial) comprehension of the visual perception increases the performance of the autonomous navigation method for learning and testing on the randomized figure-8 racetrack. Considering that the variants of experiment 1 make navigation decisions at 50 Hz, R1 learns to mem-

orize what it has seen for a time span of 0.5 s from the training on input sequences with a length of 25. In other words, R1 bases its navigation decision-making on 0.5 s videos, while F2 bases it on single images. Unlike a single frame, a video encodes the drone's latest motion sequence relative to the racetrack and the environment. This information could, for example, resolve the ambiguities stemming from the intersections of the figure-8 racetrack or render the decision-making more robust against single outlier frames that differ significantly from the aggregated training data.

The second recurrent variant of experiment 1 (R2) differs from R1 in the additional use of the optional inputs (i.e., time increment of RGB images, IMU data with time increment). R2 completed the imitation learning process with $\sim 19\%$ more rollouts and $\sim 15\%$ more training samples than R1. In $\sim 15\%$ more epochs, R2 achieved a $\sim 33\%$ lower final training loss but a $\sim 679\%$ higher final validation loss than R1. This results in a $\sim 1063\%$ higher, validation-training loss ratio than R1, while both variants have a resultant dropout probability of 50% during training. The lower training loss of R2 suggests that the optional inputs are related to the navigation decisions and R2 learns to use these relations. For example, R2 could have learned to weigh the incoming images with their time increment information or to refine the drone's motion history with the incoming IMU data. However, the by far larger validation-training loss ratio shows that R2 heavily overfits its aggregated training data, which reflects in the race test performance. The RCS of R2 is ~ 20 percentage points lower than R1 for maximum speeds from 5 to 9 m/s. The reason for the overfitting could be that combining both the visual and physical features in the data from the drone's onboard camera and IMU, respectively, increases the variance of the decision-making basis enormously. As a result, R2 would require much more training data than R1 for generalization. R2 could overcome its overfitting on more training data, whereby it possibly could outperform R1 in the race tests.

The third recurrent variant of experiment 1 (R3) differs from R1 in the fact that the Resnet14 was pretrained and only partly trainable in the learning process. This was a successful attempt to speed up the time of a training epoch. However, it made the training less effective, whereby the imitation learning process took more rollouts and epochs, which canceled the time advantage. R3 completed the imitation learning process with $\sim 56\%$ more rollouts and $\sim 55\%$ more training samples than R1. In $\sim 57\%$ more epochs, R3 achieved a $\sim 641\%$ higher final training loss and a $\sim 144\%$ higher final validation loss than R1. This results in a $\sim 66\%$ lower validation-training loss ratio than R1, while both variants have a resultant dropout probability of 50% during training. R3 cannot benefit from the low ratio and transfers the significantly higher losses to the race tests, where it performs ~ 30 percentage points worse than R1 at the maximum speeds from 5 to 9 m/s. The very little validation-training loss ratio of R3 indicates that R3 does basically not overfit. Looking at the loss trends of R3, one can see that the trainings with the CNN submodule trainable drastically reduce the losses but also

increase the validation-training loss ratio, while the trainings with the CNN module not trainable slightly reduce the losses and also the validation-training loss ratio. This could mean that the overfitting of the other variants of experiment 1 mainly stem from the CNN submodule.

At a maximum speed of 9 m/s, all three recurrent variants perform better than the feedforward variants. It is therefore likely that the higher the speed, the more beneficial the memory capabilities become for the navigation decision-making. One explanation could be that higher speeds require higher accelerations, which in turn require the drone to lean more into turns. Therewith, the drone's onboard camera could produce more outlier images and the above mentioned ability to recall past images to compensate for outlier images would thus be even more valuable. Another explanation could be that the memory time span is fixed at 0.5 s for the recurrent variants. The higher the speed, the further the memory expands spatially. This could make the memory more meaningful in terms of the drone's motion sequence in the racetrack.

Experiment 2 varies R1 from experiment 1 with respect to the number of GRU layers in order to investigate the impact of the GRU submodule's depth on the performance of a variant. Because of the high time expenditure of the imitation learning process, the variants of experiment 2 only train and validate with supervised learning on the final dataset aggregated by R1 in experiment 1. All variants trained for 200 epochs, whereby the training losses did not fully converge while the validation losses almost converged. The number of GRU layers had only a negligible influence on the epoch time, which is likely dominated by the loading of the sequential training data from disk. The fact that the ten-layer GRU variant (R1-10x64) has a $\sim 77\%$ higher training but a $\sim 35\%$ lower validation loss than the other multi-layer GRU variants (R1-2x64, R1-3x64 and R1-5x64) shows that the formula used to calculate the single application dropout probability for a resultant dropout probability of 50% is only an approximation. The single-layer GRU variant (R1-1x64) falls out of this comparison as it has no dropout. At the race tests, R1-1x64, R1-2x64, R1-3x64 and R1-5x64 performed equally well. They completed the racetrack at $\sim 85 - 100\%$ for maximum speeds of 4 to 8 m/s, at $\sim 65-85\%$ for 9 m/s and at $\sim 5-25\%$ for 10 m/s. R1-3x64 and R1-5x64 exhibit a slightly better performance at higher maximum speeds of 9 and 10 m/s. R1-10x64 performed significantly worse, with ~ 20 percentage points fewer over the tested maximum speeds. Compared to the equally configured but less trained out R1 in experiment 1, R1-3x64 achieved a $\sim 41\%$ lower final training and a $\sim 16\%$ higher final validation loss and performed slightly better over all tested maximum speeds. For the comparisons of R1-10x64 vs. R1-1x64, R1-2x64, R1-3x64 and R1-5x64 as well as R1 vs. R1-3x64, the worse performing variant has a higher final training loss and a lower final validation loss. The training loss, thus, seems to be more important for the race test performance than the validation loss. One reason for this could be the way data is aggregated in the imitation learning process. The training and the validation dataset are basically

a collection of mistakes made in the learning rollouts. The fully trained variant likely avoids a lot of drone states, for which the validation dataset provides a correct navigation decision, in the first place. An important insight from experiment 2 is that the further training of a variant, after it completed the imitation learning process, likely enhances the race test performance. While experiment 2 cannot make clear statements, whether more GRU layers are generally more capable in the autonomous navigation method, it suggests the use of R1-3x64 and R1-5x64 over R1-1x64 and R1-2x64. However, the worst performing R1-10x64 could leverage its greater depth in the imitation learning process to outperform the other variants.

Experiment 3 focuses on the learning ability of a variant for generalization to the randomized gap racetrack in simulation environments seen an unseen in the imitation learning process. The layout of the gap racetrack places two demands on the autonomous navigation method, i.e., to fly through left or right turns and through the gap. The gap is extremely challenging because when flying through it, there is intermediately no gate in the drone's field of view (FOV). A feedforward ANN module could only learn to pass the gap by using visual information in seen environments. A recurrent ANN module could also learn to pass the gap in unseen environments by recalling past visual information from its memory. Before the drone enters the gap, both gates surrounding the gap appear in the drone's FOV. As soon as the drone enters the gap and the FOV becomes gate-free, the ANN module could implicitly localize the next gate relative to the drone from its memory in order to make meaningful navigation decisions towards that gate. Even if the memory time span is too short to cover the entire time span in which the FOV is gate-free, after initializing the left or right turn in the covered time span, the drone could have the memory-based awareness that it is flying a left or right turn and could maintain the turn for the uncovered time span until the next gate reappears in the FOV.

Experiment 3 studies a feedforward variant (E3F) and a recurrent variant (E3R). E3F completed the imitation learning process with $\sim 35\%$ fewer rollouts and $\sim 37\%$ fewer training samples than E3R. Moreover, E3F trained for $\sim 33\%$ fewer epochs and achieved a $\sim 59\%$ lower final training loss than E3R. The better learning performance of E3F compared to E3R is in contradiction with the results of experiment 1, which could be explained by the fact that the FC submodule of E3F has $\sim 818\%$ more trainable parameters than the GRU submodule E3R. In experiment 3, unlike experiment 1 conducted later, no attention was paid to ensure that the FC and GRU submodules of the variants were approximately equal in the number of trainable parameters. Despite its worse learning performance, E3R outperforms E3F in the race tests in seen environments by $\sim 5 - 40\%$ percentage points over the tested maximum speeds. In unseen environments, both variants failed the race tests. The performance difference in seen and unseen environments suggests that both variants learned to use visual information in the environment to pass the gap. The yet better race test performance in the seen

environments of E3R compared to E3F is consistent with the results and their interpretations of experiment 1. Overall, experiment 3 failed to create the test conditions that would force the use of memory to navigate through the intermediate target loss of the gap.

Experiment 4 varies E3R from experiment 3 with respect to the input sequence length of the training samples and the size of the RGB images of the training samples and incoming at rollout in order to investigate their effects on performance. The variants of experiment 4 share the same learning capabilities, because their ANN modules are configured the same. However, the variants are provided with different resources to learn from. Like experiment 2, the variants of experiment 4 trained with supervised learning to save time. Each variant trained on its own training dataset, that had been rebuilt from the raw records of E3R with the corresponding sequence length and image size of that variant. All variants trained until near convergence. For variants of the same image size, the longer the sequence length, the lower the training loss of the variant. This substantiates the interpretation from experiment 1 that navigation decisions are temporally connected to past sensor data and that these connections are learnable by recurrent variants. For the range of investigated sequence lengths, the longer the memory time span, the more the variant can learn. For variants of the same sequence length, the larger the image size, the lower the training loss. This shows that a more detailed, visual observation is a stronger resource to learn the navigation decision-making. The race test performances of the variants are roughly the same, whereby lower training losses do not reflect in higher performances. The variants with the smaller image size perform slightly better in unseen environments than the variants with the bigger image size. A reason for this could be that, the ratio of spatial to temporal information in the training samples decreases with the image size, which favors the learning of temporal before spatial features, which are necessary to pass the gap in unseen environments as mentioned above.

Chapter 6

Conclusion and Future Work

6.1 Summary

Chapter 2 provided background information on imitation learning with dataset aggregation and the gated recurrent unit (GRU). The first section presented a definition of the general imitation learning problem, the most intuitive approach to it, named behavioral cloning, and the more advanced approach of dataset aggregation, which is applied in the experiments of this thesis. The second section introduced the class of recurrent neural networks (RNN) and the special RNN architecture of the GRU, including its state and gating mechanisms at inference that allow for temporal comprehension and how it trains with backpropagation through time.

Chapter 3 presented the autonomous navigation method of the baseline work, which is used for the experiments of this thesis. The first section introduced the three reference systems and their transformations applied by the modules of the method. The second section presented the ANN module, which has the function of making navigation decisions based on the RGB images from the drone's onboard camera. The ANN module, comprising the CNN, CAT, FC, GRU and HEAD submodules, is a modularized version of the ANN of the baseline work. It additionally integrates the CAT submodule, which extends the decision-making basis with the optional inputs, and the GRU submodule, which extends the decision-making capabilities with temporal comprehension. The third section presented the planning module of the method, which has the function of computing local trajectories based on the navigation decisions made by the ANN module or, possibly in the imitation learning process, the expert system. The fourth section presented the control stack of the method, which has the function of computing the drone's motor inputs to track the local trajectories computed by the planning module. The fifth section presented the expert system, which in the rollouts of the imitation learning process, intervenes and generates a labeled training sample whenever the ANN module makes a poor navigation decision.

Chapter 4 presented the four simulated experiments of this thesis, which studied different ANN module variants regarding different aspects. The first section presented the implementation and the configuration options of the simulation. The second section presented the process and the configuration options of the imitation learning with dataset aggregation. The third section presented the process and the configuration options of the race tests. The following four sections presented the configurations of the four conducted experiments. Experiment 1 studied two feedforward and three recurrent ANN module variants that trained with imitation learning on the randomized figure-8 racetrack in a single simulation environment and performed race tests in the same setting. Experiment 2 studied several recurrent ANN module variants, which are configured like the best performing recurrent variant of experiment 1 but differ in their numbers of GRU layers. The variants trained on the dataset aggregated by that best performing variant and performed the same race tests as in experiment 1. Experiment 3 studies a feedforward and a recurrent ANN module variant that trained with imitation learning on the randomized gap racetrack in several simulation environments. The variants performed race tests in the same simulation environments and in simulation environments unseen in the learning process. Experiment 4 studied several recurrent ANN module variants, which are configured like the better performing recurrent variant of experiment 3, but differ in the input sequence length of the training samples and the input RGB image size. The variants trained on the dataset aggregated by that better performing variant and performed the same race tests as in experiment 3.

Chapter 5 presented and discussed the experimental results of this thesis. In experiment 1, both feedforward variants stalled in the imitation learning process, while all three recurrent variants could complete it. The first feedforward variant, which represents the ANN used in the baseline work, performed by far the most rollouts and aggregated by far the most data. The second feedforward variant, which integrates a more complex CNN submodule than the first feedforward variant, performed the second most rollouts and aggregated the second most data. The first recurrent variant, which is the recurrent counterpart to the second feedforward variant, performed the least rollouts and aggregated the least data. The second recurrent variant, that differs from the first recurrent variant by using additional inputs, performed the second fewest rollouts and aggregated the second least data. The third recurrent variant, that differs from the first recurrent variant by the only partly trainable CNN submodule, performed the third fewest rollouts and aggregated the third least data. Both feedforward variants achieved roughly the same final training and validation losses, which are higher than the final losses of the three recurrent variants. The first recurrent variant has the fewest final validation loss, the second recurrent variant has the fewest final training loss and the highest final validation-training loss ratio, and the third recurrent variant has the fewest final validation-training loss ratio. In the race tests, the first feedforward variant performed by far the worst. The second feedforward variant performed substantially better. The

first recurrent variant performed by far the best. Both the second and the third recurrent variant perform worse than the first recurrent variant. In experiment 2, the single-layer GRU variant achieves the lowest training and the highest validation loss. The ten-layer GRU variant achieves the highest training and the lowest validation loss. The variants in between with 2, 3 and 5 GRU layers achieve roughly the same training and validation losses. All variants perform similarly well in the race tests except for the ten-layer GRU variant, which performs significantly worse. In experiment 3, the feedforward variant completed the imitation learning process with fewer rollouts, less aggregated data and a lower training loss than the recurrent variant. In environments seen in the imitation learning process, the recurrent variant clearly outperforms the feedforward variant in the race tests. In unseen environments, both variants failed the race tests. In experiment 4, the variants with the larger image size achieve fewer training losses than the variants with the smaller image size. Moreover, the variants with longer input sequence lengths achieve fewer training losses than the variants with shorter input sequence lengths. In seen environments, all variants perform roughly the same in the race tests. In unseen environments, the variants with the smaller image size performed slightly better than the variants with the larger image size.

6.2 Conclusion

This thesis was motivated by the consideration that humans, when approaching objects or avoiding obstacles in their immediate environment, rely primarily on their visual perception of their surroundings. Their visual perception is not limited to what is currently in their field of view, but also extends to their memory of what they have already seen. They can link this memory to their often subconscious decisions about how to move to reach a goal. This allows them to incorporate, for example, their own motion history or their estimation and anticipation of how objects move or are likely to move into their decision-making. State-of-the-art autonomous navigation methods for drones employ CNNs to derive navigation decisions from visual sensor data, thereby basing decision-making on a high spatial understanding of the environment currently in the drone's field of view. The objective of this thesis was to investigate the hypothesis that the combined spatial and temporal comprehension of human-like navigation is beneficial for the simplified navigation task of autonomous drone racing. The approach to this goal was to use the autonomous drone racing method of Kaufmann et al. [31] as a baseline, extend the baseline's ANN module with the GRU architecture and conduct simulated experiments to compare the performance of feedforward and recurrent ANN module variants.

The experimental results largely support the investigated hypothesis. In experiment 1 and 3, the recurrent ANN module variant significantly outperformed its feedforward counterpart in the race tests. This is especially remarkable for experiment 3, where the recurrent variant has significantly less trainable parameters. In experiment 1, where the

variants have comparable numbers of trainable parameters, the recurrent variants significantly outperformed the feedforward variants in the imitation learning process, which is accompanied by significantly fewer rollouts and aggregated data as well as significantly lower training and validation losses. The lower training and validation losses combined with the better or equal race test performance of the recurrent variants of experiment 1 showed that, first, the data aggregated by the recurrent variants, although less, is more comprehensive, suggesting that navigation decisions are indeed temporally connected to past visual observations from the drone's onboard camera and second, that the recurrent variant can learn to leverage these underlying temporal connections for a more accurate and more generalizing navigation decision-making. In contrast, the feedforward variant, which is by design unable to learn temporal connections, requires more data for a worse or equal race test performance, likely because the absence of temporal comprehension makes it less robust against intermediate ambiguities of the racetrack and outlying visual observations not represented by the aggregated data. Considering the fewer learning rollouts required, the recurrent variants are much more advantageous for real-world experiments where rollouts are more time-consuming, expensive and risky than in simulation. In experiment 3, a better performance of the recurrent variant on the gap racetrack type in unseen environments could have explicitly substantiated the hypothesis. However, the simulation configuration of the experiment allowed both variants to evade by learning visual clues in the environment. Experiments 2 and 4, where the variants, due to time concerns, trained with supervised learning, showed that lower training and validation losses do not automatically lead to better performance in race tests. This emphasizes how important it is for an imitation learning problem that the state distribution in the aggregated data resembles the state distribution at rollout. The results from both experiments would probably substantially differ if they trained the variants in an imitation learning process. However, the fact the variants could fit the pre-collected data more accurately, the longer the input sequence length of the data, again emphasizes the usability of temporal connections in the navigation decision-making.

None of the examined variants could keep up with the strong experimental results of the baseline work. The reason for this is likely that first, the experiments in the baseline work used another racetrack layout, hich is less challenging to the autonomous navigation method, and second, the simulation was non-photorealistic, which simplified the extraction of visual features from the less detailed images. A more stringent imitation learning configuration, resulting in more aggregated data, could achieve more comparable results. However, this would also have made the imitation learning processes even more time-intensive. A disadvantage of the recurrent variants is that the training epochs on sequential data are much more time-consuming than on non-sequential data. Even if the training epochs of the recurrent variants are more effective, whereby the learning process requires fewer epochs per rollout, the imitation learning process is longer than for feedforward variants. The experiments were only conducted in simulation. It re-

mains open if the recurrent variants could outperform the feedforward variants also in the real world in the light of greater visual detailedness and stronger disturbances. Moreover, problems could arise with a too long inference time of the variants running on a drone with restricted computational power. However, the number of MAC operations and the observations in simulation indicated that the feedforward and recurrent variants have roughly the same inference time. A major drawback of this thesis is that the experimental design is somewhat unstructured, since the experiments naturally emerged in the debugging process when implementing the baseline method, the recurrent variants and the learning process. Many preliminary tests were conducted, in which it was yet unclear whether the baseline method and the ANN module were implemented correctly. For example, for a long time, it was yet unclear whether the recurrent variants with the different operation modes of many-to-one at training and one-to-one at rollouts can learn at all. There were a variety of user-specifiable parameters to consider, which complicated the design of the experiments.

6.3 Future Work

The simulated experiments of this thesis were conducted under the restrictions of the COVID-19 pandemic with the minimal resources of my student home office. They could show that the use of a CNN-GRU architecture significantly increases the performance of the baseline method in the imitation learning process and in the race tests. Embedded in a research environment that allows for more vibrant scientific discourse and provides more computationally powerful hardware, the following open questions could be investigated.

Experiment 1 considered only a single simulation environment and in experiment 3, the variants learned in environments in only four different scenes and failed the race tests in the environments of the fifth scene unseen in the learning process. Further experiments where the variants learn in more numerous and diverse environments could investigate whether the variants can generalize to environments unseen in the learning process and whether the feedforward and the recurrent variant differ in their generalization abilities. In experiment 3, the variants learned visual clues in the environments to navigate the gap of the racetracks. More tailored experiments, where the variants learn in a monochrome environment absent of visual clues, could investigate whether the recurrent variants can learn to navigate the gap only with the help of their memory abilities. In Experiment 2 and 4, the variants trained only with supervised learning. Further experiments could conduct these two experiments with imitation learning, which would produce more meaningful results regarding the impact of the GRU submodule's configuration and memory time span induced by the input sequence length of the training samples. The race test performances in the simulated experiments of the baseline work are much better than in the experiments of this thesis, where the test conditions

were more difficult. Further experiments with more comparable test conditions could verify or falsify the results of the baseline work or could identify reasons for the performance difference. As all experiments of this thesis were simulated, the question remains open, whether real-world experiments could reinforce the better results of the recurrent variants.

In this thesis, the GRU architecture realized the temporal comprehension in the navigation decision-making. Further experiments could investigate the deployment of other recurrent architectures, such as the more prevalent long short-term memory (LSTM) [23] or the more recent Content Adaptive Recurrent Unit (CARU) [8]. In this thesis, dataset aggregation solved the imitation learning problem, where the recurrent variants learned better than the feedforward variants. Further experiments could compare the learning performance of recurrent and feedforward variants for other types of imitation learning, such as inverse reinforcement learning. In this thesis, a single autonomous drone racing constituted the baseline. Further experiments could investigate whether extending other autonomous drone racing methods or even more general autonomous navigation methods would also benefit from the extension with temporal comprehension.

Bibliography

[1] Visual drone inspection across different industries. https://www.equinoxsdrones.com/blog/visual-drone-inspection-across-different-industries. Accessed: 2021-11-30.

[2] Lecture in principles of robot autonomy 2. *Autonomous Systems Lab of Stanford University*, 2022.

[3] Dor Abuhasira. In 2022, percepto is bringing visual inspection automation to all. https://www.equinoxsdrones.com/blog/visual-drone-inspection-across-different-industries. Accessed: 2021-11-30.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, mar 1994.

[5] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots*, 42(2):291–306, nov 2016.

[6] Mike Brookes. The matrix reference manual. 2020. URL: http://www.ee.imperial.ac.uk/hp/staff/dmb/matrix/intro.html (accessed on 08/07/2022).

[7] Gino Brunner, Bence Szebedy, Simon Tanner, and Roger Wattenhofer. The urban last mile problem: Autonomous drone delivery to your balcony. In *2019 international conference on unmanned aircraft systems (icuas)*, pages 1005–1012. IEEE, 2019.

[8] Ka-Hou Chan, Wei Ke, and Sio-Kei Im. CARU: A content-adaptive recurrent unit for the transition of hidden state in NLP. In *Neural Information Processing*, pages 693–703. Springer International Publishing, 2020.

[9] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. September 2014.

[10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[11] Titus Cieslewski, Elia Kaufmann, and Davide Scaramuzza. Rapid exploration with multi-rotors: A frontier selection method for high speed flight. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2017.

[12] Pamela Cohn, Alastair Green, Meredith Langstaff, and Melanie Roller. Commercial drones are here: The future of unmanned aerial systems. `https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/commercial-drones-are-here-the-future-of-unmanned-aerial-systems`. Accessed: 2021-11-30.

[13] Jason Dunn. Drone use taking flight on small farms. `https://www.munichre.com/topics-online/en/mobility-and-transport/drone-use-taking-flight-on-small-farms.html`. Accessed: 2021-11-30.

[14] Matthias Faessler, Davide Falanga, and Davide Scaramuzza. Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight. *IEEE Robotics and Automation Letters*, 2(2):476–482, apr 2017.

[15] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robotics and Automation Letters*, 3(2):620–626, apr 2018.

[16] Mikel L. Forcada and Rafael C. Carrasco. Learning the initial state of a second-order recurrent neural network during regular-language inference. *Neural Computation*, 7(5):923–930, sep 1995.

[17] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. RotorS—a modular gazebo MAV simulator framework. In *Studies in Computational Intelligence*, pages 595–625. Springer International Publishing, 2016.

[18] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, oct 2017.

[19] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *Lecture Notes in Computer Science*, pages 195–201. Springer Berlin Heidelberg, 1995.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. December 2015.

[21] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. July 2012.

[22] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, nov 1997.

[24] Xiaolin Hu and P. Balasubramaniam. *Recurrent neural networks*. InTech, 2008.

[25] IBM Cloud Education. Recurrent neural networks. *IBM Cloud Learn Hub*, 2020. URL: `https://www.ibm.com/cloud/learn/recurrent-neural-networks` (accessed on 04/07/2022).

[26] P Jackson. Introduction to expert systems. 1 1986.

[27] Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, 3(3):2539–2544, jul 2018.

[28] Sunggoo Jung, Hanseob Lee, Sunyou Hwang, and David Hyunchul Shim. Real time embedded system framework for autonomous drone racing using deep learning techniques. In *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*. American Institute of Aeronautics and Astronautics, jan 2018.

[29] Lukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms, 2015.

[30] Elia Kaufmann, Mathias Gehrig, Philipp Foehn, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, may 2019.

[31] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. 2018.

[32] Hoang M Le, Peter Carr, Yisong Yue, and Patrick Lucey. Data-driven ghosting using deep imitation learning. 2017.

[33] Erik Learned-Miller. Vector, matrix, and tensor derivatives. URL: `http://cs231n.stanford.edu/vecDerivs.pdf` (accessed on 08/07/2022).

[34] Minchen Li. A tutorial on backward propagation through time (bptt) in the gated recurrent unit (gru) rnn. *Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, Tech. Rep*, 2016.

[35] Yi Lin, Fei Gao, Tong Qin, Wenliang Gao, Tianbo Liu, William Wu, Zhenfei Yang, and Shaojie Shen. Autonomous aerial navigation using monocular visual-inertial fusion. *Journal of Field Robotics*, 35(1):23–51, jul 2017.

[36] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU. *IEEE Robotics and Automation Letters*, 2(2):404–411, apr 2017.

[37] Antonio Loquercio and Davide Scaramuzza. Learning to control drones in natural environments: A survey. In *e ICRA Workshop on Perception, Inference, and Learning for Joint Semantic, Geometric, and Physical Understanding*, number CONF, 2018.

[38] Michal Mazur, Adam Wisniewski, and Jeffery McMillan. *Clarity from above*. PwC Poland, May 2016.

[39] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, may 2011.

[40] Luc Le Mero, Dewei Yi, Mehrdad Dianati, and Alexandros Mouzakitis. A survey on imitation learning techniques for end-to-end autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–20, 2022.

[41] Ali A. Minai and Ronald D. Williams. On the derivatives of the sigmoid. *Neural Networks*, 6(6):845–853, jan 1993.

[42] Hyungpil Moon, Jose Martinez-Carranza, Titus Cieslewski, Matthias Faessler, Davide Falanga, Alessandro Simovic, Davide Scaramuzza, Shuo Li, Michael Ozo, Christophe De Wagter, Guido de Croon, Sunyou Hwang, Sunggoo Jung, Hyunchul Shim, Haeryang Kim, Minhyuk Park, Tsz-Chiu Au, and Si Jung Kim. Challenges and implemented technologies used in autonomous drone racing. *Intelligent Service Robotics*, 12(2):137–148, jan 2019.

[43] Mark W. Mueller, Markus Hehn, and Raffaello D'Andrea. A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification. pages 3480–3486, Tokyo, Japan, 2013. IEEE.

[44] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, oct 2015.

[45] Artem Nikolov. Introduction into imitation learning. *Institute for Computational Linguistics, Heidelberg University*, 2018.

[46] Rick Parent. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann.

[47] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[48] Raúl Rojas. The backpropagation algorithm. In *Neural Networks*, pages 149–182. Springer Berlin Heidelberg, 1996.

[49] Leticia Oyuki Rojas-Perez and Jose Martinez-Carranza. DeepPilot: A CNN for autonomous drone racing. *Sensors*, 20(16):4524, aug 2020.

[50] Ellen Rosen. Skies aren?t clogged with drones yet, but don?t rule them out. *The New York Times*, March 2019.

[51] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2010.

[52] D. S., Milton Abramowitz, and Irene A. Stegun. Handbook of mathematical functions with formulas, graphs, and mathematical tables. *Mathematics of Computation*, 20(93):167, jan 1966.

[53] Inkyu Sa, Mina Kamel, Michael Burri, Michael Bloesch, Raghav Khanna, Marija Popovic, Juan Nieto, and Roland Siegwart. Build your own visual-inertial drone: A cost-effective and open-source autonomous drone. *IEEE Robotics & Automation Magazine*, 25(1):89–103, mar 2018.

[54] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. November 2016.

[55] Davide Scaramuzza, Michael C. Achtelik, Lefteris Doitsidis, Fraundorfer Friedrich, Elias Kosmatopoulos, Agostino Martinelli, Markus W. Achtelik, Margarita Chli, Savvas Chatzichristofis, Laurent Kneip, Daniel Gurdan, Lionel Heng,

Gim Hee Lee, Simon Lynen, Marc Pollefeys, Alessandro Renzaglia, Roland Siegwart, Jan Carsten Stumpf, Petri Tanskanen, Chiara Troiani, Stephan Weiss, and Lorenz Meier. Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in GPS-denied environments. *IEEE Robotics & Automation Magazine*, 21(3):26–40, sep 2014.

[56] Jürgen Schmidhuber. The most cited neural networks all build on work done in my labs. *Jürgen Schmidhuber's AI Blog*, 2021. URL: `https://people.idsia.ch/~juergen/most-cited-neural-nets.html` (accessed on 04/07/2022).

[57] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. September 2020.

[58] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing with deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2021.

[59] Sarah Taylor, Taehwan Kim, Yisong Yue, Moshe Mahler, James Krahe, Anastasio Garcia Rodriguez, Jessica Hodgins, and Iain Matthews. A deep learning approach for generalized speech animation. *ACM Transactions on Graphics*, 36(4):1–11, jul 2017.

[60] Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Imitation learning at all levels of game-ai. In *Proceedings of the international conference on computer games, artificial intelligence, design and education*, volume 5, 2004.

[61] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing, 2017.

[62] Yisong Yue and Hoang Le. Imitation learning tutorial. *Tutorial at ICML*, 2018:3, 2018.

[63] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2017.

[64] Brian D. Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J. Andrew Bagnell, Martial Hebert, Anind K. Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, oct 2009.

[65] Hans-Georg Zimmermann, Christoph Tietz, and Ralph Grothmann. Forecasting with recurrent neural networks: 12 tricks. In *Lecture Notes in Computer Science*, pages 687–707. Springer Berlin Heidelberg, 2012.

Appendices

Appendix A: Thesis Implementation

For this thesis, I implemented the autonomous navigation method and the simulated experiments as a ROS package named forgetful_drone[1].

[1] `https://github.com/fcvm/forgetful_drone.git`, accessed on October 5, 2022

Appendix B: Racetrack Randomization and Redirection

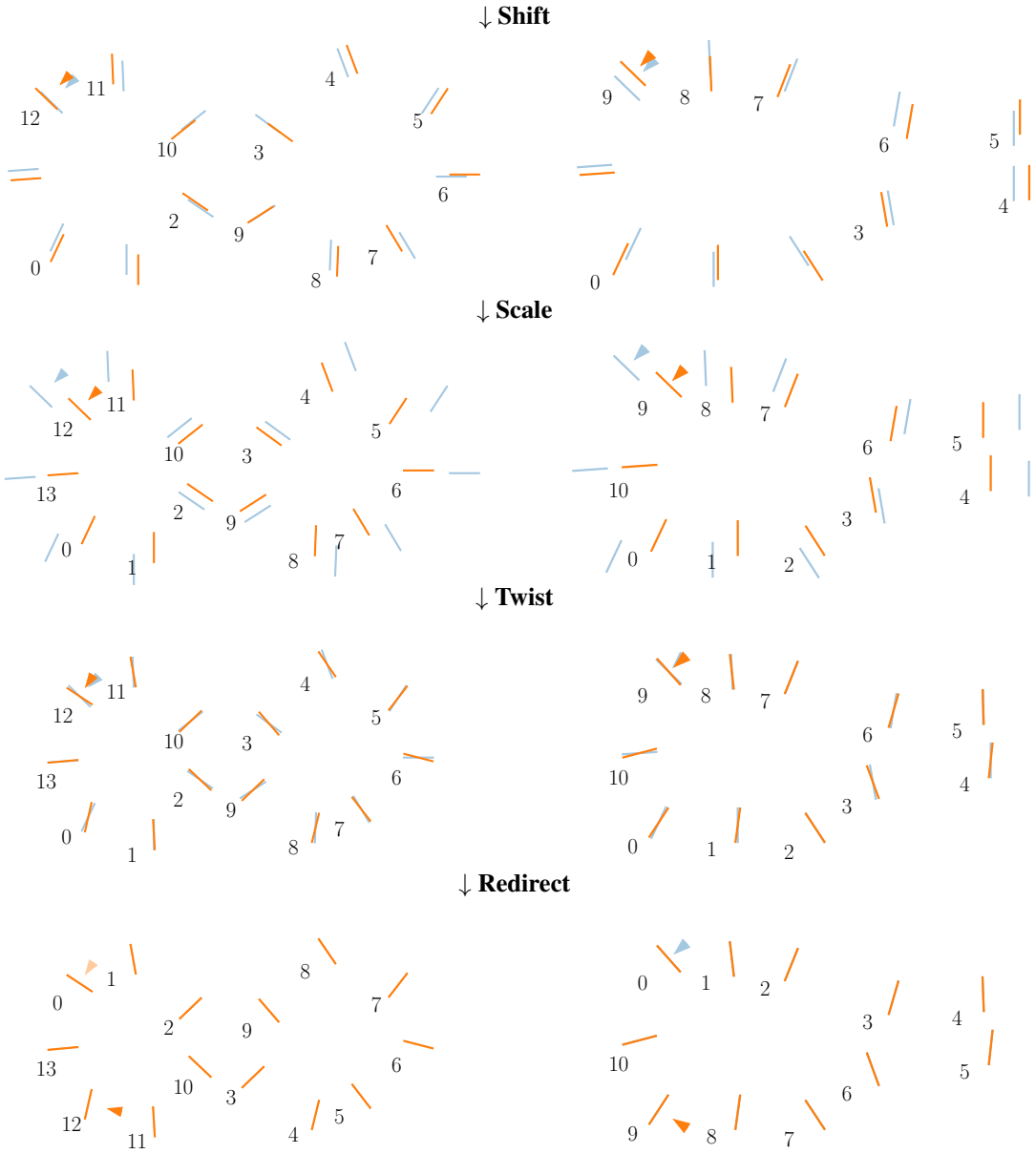


Figure 6.1: Racetrack randomization and redirection for the figure-8 (left) and the gap (right) racetrack types. In the xy-plane, the start pose of the drone (arrow) and the gate poses (numbered bars) are shown before (low-opacity blue) and after (orange) the randomizing processing steps Shift, Scale and Twist and the Redirect processing step.