Technische Universität Berlin

A|O|T
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

DOCUMENT BUILD DATE: September 25, 2022
DOCUMENT STATUS: Beta

Temporal comprehension in autonomous drone racing: infer navigation decisions from current and past raw sensor data with a recurrent convolutional neural network

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

DELETEME: An abstract is a teaser for your work. You try to convince a reader that it is worth reading your work. Normally, it makes to structure you abstract in this way:

- one paragraph on the motivation to your topic
- one paragraph on what approach you have chosen
- and one paragraph on your results which may be presented in comparison to other approaches that try to solve the same or a similar problem.

Abstract should not exceed one page (aubrey's opinion)

Zusammenfassung

DELETEME: translate to German to Englisch or vice-versa.

Acknowledgements

DELETEME: Thank you for the music, the songs I am singing

Yacine Zahidi: Paris Pytorch Lukas Kilian: Computer Unity Administratives HU: RÃ¼cken freigehalten, immer unterstÃ¼tzt

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 [12]. Today, drones are already standard in aerial inspection services [11, 1, 3]. In the near future, the areas of infrastructure, transport, insurance, media and entertainment, telecommunication, agriculture, safety and mining are considered to be particularly promising [36]. In contrast, real breakthroughs of more visionary applications, such as drone delivery or drone taxis are, if at all, only expected in the more distant future [48]. Reasons for this are strict regulations, skeptical public attitudes and technical problems [48]. With regard to 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 [35] 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. Simple 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. [42] Path planning algorithms (e.g., [5], [10]) are applied 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., [33], [53], [51], [34]). Theoretically, these apporaches can be deployed

in uncontrolled environments with unknown obstacles and other agents. Practically, the mapping coerces global consistency which likely makes the computation too complex to be run in real-time 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., visual, range, IMU) and the decisions making of an ANN. These methods can be further subdivided by their way of learning into reinforcement learning (RL) and imitation learning (IL). RL has the advantage that they are not affected by the "distributional mismatch" [2] between training and testing as the ANN learns with "trial-and-error" [52] from direct environmental interaction. However, this comes with the expense of a substantially higher sample complexity compared to IL. [61] 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. [52]

A popular research playground is autonomous drone racing (e.g., [40], [25], [56]) due to the controlled flight environment, the straightforward definition of goals and the measureability 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., [47], [28], [26]). 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 safely navigate 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 to observe 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 a recurrent neural networks in autonomous drone racing.

1.2 Approach and Goals

This thesis aims to investigate the question whether temporal comprehension induced by a recurrent neural network can be beneficial for the autonomous navigation in drone racing. To do this, the thesis uses the vision-based autonomous navigation method proposed by Kaufmann et al. [29] 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 artifical 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, thereby enabling the baseline method to spatially comprehend what is in the field of 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 would be made based 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, is expected to improve the racing performance. This hypothesis is investigated in simulated drone races. Different variants of ANNs are tested for their robustness 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 intermediately leaves the FOV of the drone's camera (e.g., due to 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

This thesis is structured as follows. Chapter 2 provides background information on the thesis topic. This includes the machine learning field of imitation learning, in particular with dataset aggregation, and recurrent neural networks, in particular the gated recurrent unit. Chapter 3 presents the applied autonomous drone racing navigation method, which consists of the ANN, planning and control submodules. Special attention is paid here to the ANN submodule, as it is the part that enables temporal comprehension and therewith distinguishes this thesis from the original method. Chapter 4 presents the experiments of the thesis. Racing tests for the navigation method with different ANN submodule variants are carried out in simulation. Thereby, the performance of the variants are measured. Chapter 5 evaluates the results of the racing experiments. The focus lies on the question, whether temporal comprehension is beneficial to the racing performance. Chapter 6 is the conclusion of this thesis. At last, Chapter 6.3 gives additional related information on the topic of this thesis.

Chapter 2

Background

DELETEME: This chapter will cover all of your background information and related work. Background and related work are directly related to your thesis. Please do not place irrelevant content here which is a common mistake. Citing will be handled in the appendices.

DELETEME: Background represents underlaying knowledge that is required to understand your work. The expected knowledge level of your readers can be set to the one of a bachelor or master student who just finished his studies (depending on what kind of thesis you are writing). This means that you do not need to describe how computers work, unless your thesis topic is about this. Everything that an avarage alumni from your field of studies should now does not need to be described. It turn, background information that is very complex and content-wise very near to you problem, can be placed in the main parts. Everyting else should be written here. Note: it is important to connect each presented topic to your thesis. E.g. if you present the ISO/OSI layer model you should also write that this is needed to understand the protocols you plan to develop in the main parts.

DELETEME: Related work respresents results from work that handled the same or a similar problem that you are addressing. This work might have used a different approach or might not have been that successful. Finding a paper / work that solved your problem in the same way you were planning to do is not good and you should contact your supervizor for solving this issue. Again, each paper / work has to be connected to your approach: other papers might have not chosen an optimal solution; they might not have been taking care of essential aspects; they might have chosen a different approach and you believe, yours will work better ...

2.1 Topic 1

2.2 Imitation Learning with Dataset Aggregation

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. It is a potential solution to learning problems where it is easier to demonstrate the desired behavior of attaining a goal than to define intermediate goals and rewards (as in supervised and reinforcement learning, respectively) that lead to the desired behavior. In terms of supervision, imitation learning is therefore located between supervised learning and reinforcement learning. [43] Imitation learning tends to apply successfully on tasks related to human intent/behaviour, e.g., pedestrian avoidance [62], speech animation [57], sport analytics [30] and gaming [58]. With respect to planning and control in autonomous navigation, imitation learning has become state-of-the-art. [38] The methods of imitation learning can be divided by their approch into direct policy learning and inverse reinforcement learning. [2] Direct policy learning optimizes a policy class to approximate the expert's behaviour 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.

In this thesis, the training of the ANN module of the autonomous navigation method (see section 3.2) is considered an imitation learning problem. In the experiments (see section 4.2), direct policy learning with dataset aggregation is applied to solve this problem. Therefore this section introduces the core concept of dataset aggregation, which was proposed as DAgger by Ross et al. [49]. Prior to this, the general imiation learning problem is defined and behavioural cloning, the simplest form of direct policy learning is presented.

Problem definition

An imitation learning problem includes: a system, an expert, a policy class, a loss function, and a learning algorithm. The goal is to find the policy within the policy class that approximates the expert's decision making as closely as possible. [60]

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 possible system states and actions, respectively. The system is considered a MDP (Markov decision process) whose initial state is sampled from the distribution $p(\underline{x}_0)$. The system dynamics can be described with a probabilistic transition model, which provides 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. This decision making process is described by the expert policy which maps system states to expert actions

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

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

The policy class defines the search space the optimal policy, i.e., the policy that most closely imitates the expert policy. In the following, the policy class is considered to be an ANN (artifical neural network)

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

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

Both, the expert policy and members of the policy class, can be rolled 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 infered 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 state distribution of a policy rollout, i.e., the probability that a state is visited during the rollout of the policy and becomes therewith part of the corresponding trajectory, is induced by the policy itself. It is calculated by averaging 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 quantifies how closely a policy π imitates the expert policy π^*. For a state, it measures the deviation of the action predicted by the general policy from the action demonstrated by the expert

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

The learning algorithm finds the policy $\hat{\pi}^*$ within the policy class that is closest to the expert policy. For this purpose, the algorithm minimizes the total imitation loss evaluated for states sampled from the distribution induced by the identified policy itself. In machine learning, the optimization is realized with the training of the ANN, whereby its trainable parameters are updated according 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 corresponds to a "non-i.i.d. supervised learning problem" [49]. Note that, as generally the system dynamics are unknown or too complex, the training data cannot be computed to be independent and identically distributed (i.i.d.) across the state space $\mathcal{X}$. In fact, the only way to generate training data is to sample it during policy rollouts. General optimization learning takes the most reliable, available option of generating training data from rollouts of the identified policy itself. This has the advantage, that the state distributions at training and testing (rollout) do not differ and thus, a low training loss is likely to transfer to good test performance. On the downside, the interdependence of the ANN and the training state distribution renders the optimization to be non-convex and thus complex in contrast to common supervised learning.

Behavioural cloning

Behavioural cloning reduces the imitation learning problem to a single standard supervised learning problem on a precollected training dataset of expert trajectories

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

Each trajectory is the product of an expert policy rollout and contains the thereby demonstrated state-action pairs. The state distribution of an expert rollout is

$$p\left(\underline{x}|\pi^*\right) = \frac{1}{T+1} \sum_{t=0}^{T} p\left(\underline{x}_t|\pi^*\right). \tag{2.10}$$

The optimization problem of BC is thus to minimize the accumulated imitation loss of all demonstrations $(\underline{x}_t, \underline{u}_t^*)$ of all trajectories τ_i within the dataset D precollected

$$\begin{aligned} \hat{\pi}^* &= \underset{\underline{\theta}}{\operatorname{argmin}} \sum_{\tau_i \in D} \sum_{(\underline{x}_t, \underline{u}_t^*) \in \tau_i} L\left(\pi_{\underline{\theta}}\left(\underline{x}_t\right), \underline{u}_t^*\right) \\ &= \underset{\underline{\theta}}{\operatorname{argmin}} \sum_{\underline{x} \sim p(\underline{x}|\pi^*)} L_{\pi_{\underline{\theta}}}\left(\underline{x}\right). \end{aligned} \tag{2.11}$$

In comparison to general imitation learning, behavioural 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 iid across the state space makes it likely that the ANN is confronted with states not seen during training. As a result, a good test performance cannot be generally infered from a low training loss. This is especially true for a long task horizon T of a rollout. The above optimization can be interpreted as minimizing the one-step loss of the trained ANN along the expert trajectories. However, as the trained ANN rolls out at testing, these one-step errors, even though minimized, accumulate along the ANN trajectory and will likely cause non-negligible deviation from the expert's behaviour. In fact, on a ANN trajectory the upper bound of the error grows quadratically with the number of timesteps. [60]

DAgger
DAgger[49] is an iterative method for direct policy learning that reduces the imitation learning problem to a sequence of convex supervised learning problems:

1. Roll out the expert policy to generate a dataset. Train the initial ANN $\pi_{\underline{\theta},0}$ with behavioural cloning on that dataset.
2. For $i = 1, 2, \ldots$
   (a) Rollout the initial/intermediate ANN $\pi_{\underline{\theta},i-1}$ to generate trajectories.
   (b) Label the trajectories using the expert policy and add them to the training dataset.
   (c) Train the next intermediate ANN $\pi_{\underline{\theta},i}$ on the aggregated dataset.

In contrast to behavioural cloning, DAgger requires an interactive expert, that provides feedback on the state distributions induced by the ANN, and access on the system to execute ANN rollouts. The DAgger algorithm makes sure that mistakes made by the ANN during rollout become part of the aggregated dataset. Ideally, this training converges to the optimal ANN parameters for the general IL problem.

2.3 Gated recurrent unit

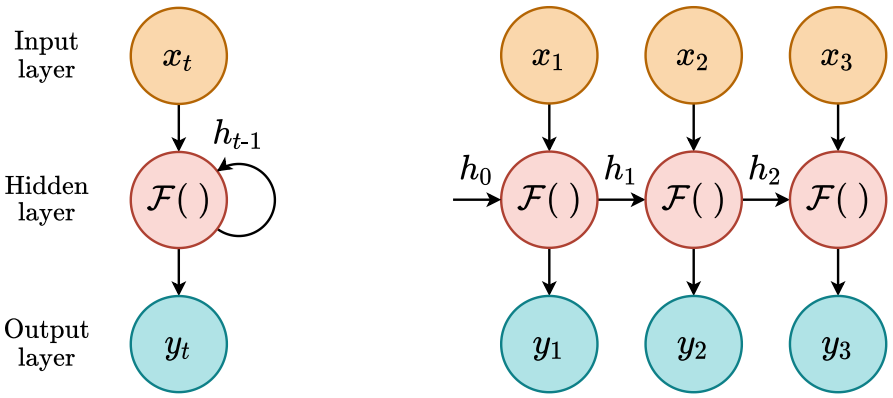
The artifical neural network (ANN) of the autonomous navigation method of this thesis integrates the PyTorch implementation[1] of the gated recurrent unit (GRU) with the objective to involve temporal comprehension in the navigation decision making. The GRU, published in 2014 by Cho et al. [8], is a variant of the ANN class of recurrent neural networks (RNN). This section shortly introduces the RNN class and justifies the design choice for the GRU in light of standard RNNs and the more prevalent RNN

[1] `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`, visited on 09/07/2022

variant, the long short-term memory (LSTM). Moreover, it provides an insight into the mechanisms of the GRU that make temporal comprehension available as well as how the GRU trains its temporal comprehension by backpropagating loss through time.

Recurrent neural networks

RNNs contrasts with classical feedforward ANNs, which forward information exclusively towards subsequent layers, by featuring dynamic properties that stem from the implementation of feedback connections [22] (see fig. 2.1a). As previously infered states re-enter the network, the output of an RNN depends not only on the current but also on prior inputs. In this sense, an RNN is qualified to process and reason on entire sequences of data points. In case of time-series data, this can be interpreted as temporal comprehension and memory [23]. RNNs train on sequential data with backpropagation through time (BPTT) [45] which is basically the application of standard backpropagation (e.g., [46]) on the unfolded representation of the RNN (see figure 2.1b). The unfolded representation exhibits a layer for each data point in the given input sequence and can be construed 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 sequences is hence prone to the vanishing gradient problem [20], which manifests itself in the premature slowdown or standstill of the convergence of the RNN. As a result, standard RNNs have difficulties learning to connect to information from inputs deep in the past [4].



(a) Folded (b) Unfolded

Figure 2.1: Folded schematic RNN with a single hidden layer that feeds the previous h_{t-1} (or initial h_0) state back to the inference at the current time step t and its time-unfolded representation when processing the input of a time-series consisting of the three data points $(x_i)_{i\in\{1,2,3\}}$.

In 1997, Hochreiter and Schmidhuber [21] introduced the long short-term memory

(LSTM), which is today's predominant [54] RNN variant. A standard LSTM layer (see, e.g., the PyTorch implementation[2]) recurrently maintains a cell and a hidden state. This means that, in addition to the current data point of the input sequence, the layer re-inputs the fed back cell and hidden state from the previous sequential step. Furthermore, the LSTM layer implements three gating mechanism that 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 on a state, controls the flow of the elements of the state 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 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 [45]. As a result, the LSTM is, compared to standard RNNs, essentially 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. [8], can be seen as a lighter version of the LSTM. It refrains from the cell state and therewith the output gate of the LSTM and maintains only a hidden state and two gating mechanisms. Therewith, the GRU has less trainable parameters than the LSTM and is less memory efficient during inference. Nevertheless, it preserves the robustness with respect to the vanishing gradient problem that affects most standard RNNs [23]. Various empirical studies show that the GRU, compared to the LSTM, performs equally well or even slightly better. Greff et al. [17] evaluated the "vanilla" LSTM and the GRU, among other LSTM variants, on speech and handwritten text recognition as well as music representation and could not detect substantial differences in performance. Chung et al. [9] applied LSTM and GRU on raw speech and polyphonic music data and empirically observed that both performed equally well. In the comparative study of Yin et al. [59], the GRU slightly outperforms the LSTM on five of seven natural language processing tasks. In the field of algorithm learning, Kaiser and Sutskever [27] achieved notedly better results with a network based on GRU than with a network based on LSTM. In consideration of these findings and the fact that the GRU has less trainable parameters and occupies less memory during inference, the GRU is chosen over the LSTM for the deployment in the ANN module of the autonomous navigation method of this thesis.

Inference

The following presents the mathematics of a single GRU layer during inference in ac-

[2] `https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html`, visited on 03/07/2022

cordance with the PyTorch implementation[3]. This includes the GRU layer's two gating mechanisms as well as its computations of the candidate 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

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

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

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

with the dimensionality N^{hidden} 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 initialized with zeros or noise [63] but can also be learned by the network, e.g., [15]. The GRU layer processes the time series included in the batch parallely and the data points of the individual time series successively. At the current time step t, the layer's input consists of 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 reasons of simplification, the following text only refers to the parallely computed, individual elements of the processed batch. However, the following equations yet explicitely refer to the entire batch.

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

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

is obtained by the mapping

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

The above mapping to the reset gate can be devided into two steps. First, the current data point and the previous hidden state are linearly transformed with the corresponding matrices of trainable weights and vectors of trainable biases

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

[3] `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`, visited on 09/07/2022

The user has the design option to disable all biases of the GRU layer. This is tantamount to set the above and the still upcoming biases of the layer to zero and consider them not trainable. Second, the standard sigmoid function [18] (see figure 2.2)

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

is applied element-wise (denoted with the accent $\odot$) on the sum of these two linear transformations. The sigmoid function forces the values of the reset gate to be in the interval between zero and one. This is characteristic for a gating mechanism since the gating of a state, which is in fact the calculation of the Hadamard product of the state and the gate, targets at only damping not amplifying or reversing the individual values of the state.

The current update gate

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

is obtained with the mapping

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

The above mapping to the update gate differs from the mapping to the reset gate only in the fact that the ocurring linear transformations rely on their seperate, trainable weights and biases

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

With the knowledge of the current reset gate, the GRU layer calculates the candidate for the current hidden state, hereinafter referred to as candidate state

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

The mapping to the candidate state

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

contains the trainable weight matrices and bias vectors

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

The above mapping to the candidate state resembles the mappings to the reset and update gate by subjecting the current data point and the previous hidden state to a seperate, biased linear transformation. It differs in two aspects. First, before the two transformations are added together, the current reset gate is applied on the transformed, previous hidden state ($\odot$ denotes the Hadamard product). The function of the reset gate is, consequently, to mitigate the contribution of the previous hidden state to the candidate state. Second, instead of the sigmoid function, the hyperbolic tangent [50] (see figure 2.2)

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

is applied element-wise on the sum of the linearly transformed data point and the gated, linearly transformed, previous hidden state. The hyperbolic tangent bounds the values of the candidate state to the interval from minus to plus one.

Finally, the GRU layer computes the current hidden state

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

on the basis of the mapping

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

The above mapping to the hidden state is basically a weighted arithmetic mean. Before the previous hidden state and the current candidate state are averaged, they are weighted by the current update gate and counter-update gate, respectively. In other words, the update gate, whose values are between zero and one, controls the element-wise percentage proportions of both states on the current hidden state. Due to the fact that the hyperbolic tangent normalizes the elements of the candidate state to the interval from minus to plus one (equation 2.24) and the values of the initial hidden state are typically initialized to be also within this interval (equation 2.14), the values of the current hidden state are also bounded to the interval from minus to plus one.

Backpropagation through time

The following presents the application of backpropagation through time on a single integrated GRU layer operating in many-to-one mode in order to calculate the gradients of the loss with respect to the GRU layer's trainable parameters. During training, these gradients are required by gradient-based methods which update the trainable parameters with the target to minimize the loss.

Let a single GRU layer, integrated into any superior ANN, operate in many-to-one mode. With biases enabled, the set of all structures that contain trainable parameters of

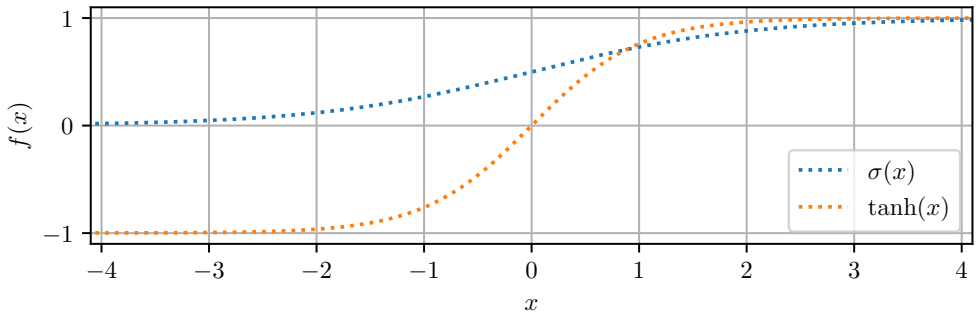


Figure 2.2: The non-linear activation functions deployed within a standard GRU. The standard sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$ normalizes the values of the reset and update gate to the interval from zero to one (equation 2.17 and 2.21). The hyperbolic tangent $\tanh(x) = (e^{x} - e^{-x})(e^{x} + e^{-x})^{-1}$ normalizes the values of the candidate and, consequently, the hidden state to the interval from minus to plus one (equation 2.24 and 2.28).

the GRU layer is aggregated from equation 2.18, 2.22 and 2.25

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

The addition of the sizes of the structurs in the set Θ yields the total number of trainable parameters of the GRU layer

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

At a single inference, the GRU layer receives a batch of sequences of feature vectors from the previous layer of the ANN

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

The GRU layer in many-to-one mode maps each incoming batch of sequences to a batch of single outputs, whereby a single output is the hidden state lastly infered from a sequence

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

The time-unfolded computation graph of the integrated GRU layer for a single batch element is shown in figure 2.3. The batches of single GRU outputs are forwarded to the subsequent layer of the ANN. Whenever a batch has passed all the output layer of the ANN, the loss L of the batch is computed by averaging the losses of the individual

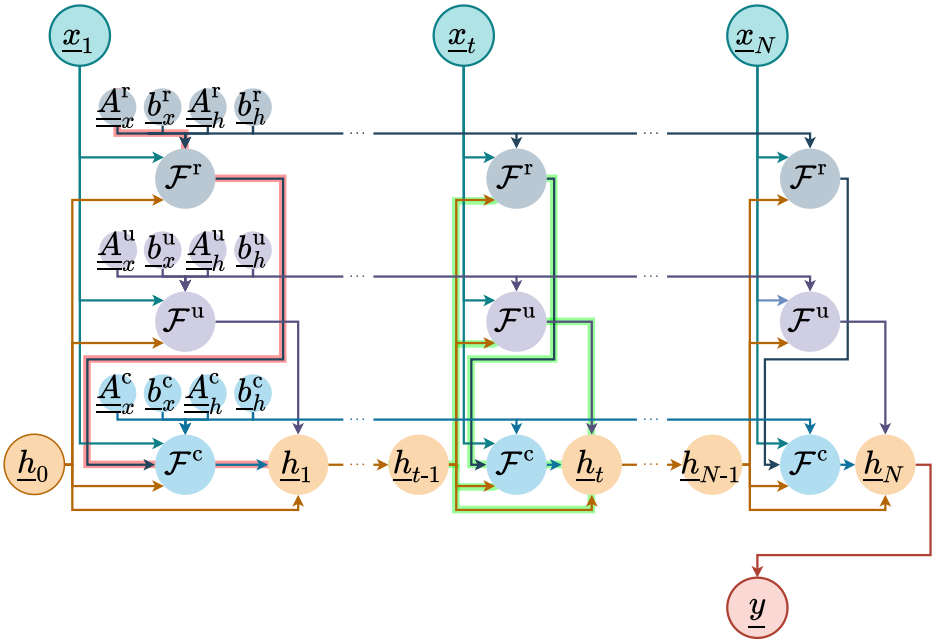


Figure 2.3: Time-unfolded computation graph of a GRU layer operating in many-to-one mode. The input sequence $(x_t)_{t\in\{1,\dots N\}}$ is mapped to the single output $\underline{y}$ which is equal to the lastly inferred, hidden state $\underline{h}_N$. Equation 2.17, 2.21 and 2.24 define the mapping to the reset gate $\mathcal{F}^{\mathrm{r}}$, update gate $\mathcal{F}^{\mathrm{u}}$ and candidate state $\mathcal{F}^{\mathrm{c}}$, respectively. The hidden states $(h_t)_{t\in\{1,\dots N\}}$ are obtained with the mapping $\mathcal{F}^{\mathrm{h}}$ defined by equation 2.28 whereas the initial hidden state $\underline{h}_0$ (equation 2.14) is defined by the user. The trainable parameters $\underline{\underline{A}}^{\square}_{\square}, \underline{b}^{\square}_{\square}$ of the GRU layer are shared across all unfolded time steps. The backpropagation path of the intra-gradient with respect to $\underline{\underline{A}}^{\mathrm{r}}_x$ (equation 2.36) is highlighted in red for the first time step $t = 1$. The backpropagation paths of the inter-gradient (equation 2.39) is highlighted in green for the time step t.

elements of the batch. Because the seperate losses depend on their corresponding single GRU output, the loss of the batch is a function of all of these outputs.

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

Gradient-based methods update the trainable parameters of the ANN with the goal to minimize the loss of the batch. For this, they require the knwoledge of the gradients of the 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 [32], the gradient of the batch loss with respect to a single element $\theta \in \Theta$ of the set of structures containing trainable parameters (equation 2.29) is computed based on backpropagation through time

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

(1) Loss of a batch (equ. 2.33)

(2) Chain rule

(3) Backpropagation through time (gradient $\widehat{\square}$ considers previous hidden states constant)

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

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 only covers the backpropagation path from the hidden state $\underline{h}_{j,i}$ to the parameter structure θ within the same time step j. In order to compute the full gradient covering all

backpropagation paths to θ, all intra-backpropagation paths are connected through time to the GRU layer's output $\underline{y}_i$ by multiplying the intra-gradients with their corresponding chain of time step inter-gradients $\partial \underline{h}_{k,i} / \partial \underline{h}_{k-1,i}$. For demonstration purposes, the intra-gradient with respect to $\theta = \underline{\underline{A}}^{\mathrm{r}}_{x}$ (the corresponding backpropagation path is highlighted for the first time step in figure 2.3) is examplarily derived as

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

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

(2) Sum rule of differentiation;

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

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

Hadamard product of two vectors (equ. 2.45)

with

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

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

(2) Chain rule of differentiation;

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

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

Hadamard product of two 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) Mapping to reset gate (equ. 2.17)

(2) Chain rule of differentiation;

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

(3) Derivative of sigmoid function (equ. 2.44)

For the computation 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 3-dimensional matrix whose information content is only 2-dimensional, refer to [31], for example.

The in equation 2.35 required inter-gradient (the corresponding backpropagation path is highlighted in figure 2.3) , i.e., the gradient of the current hidden state with respect to the previous hidden state, is derived as

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

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

(2) Sum rule of differentiation

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

The in equation 2.39 required gradient of the current update gate with respect to the

previous hidden state is derived as

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

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

(2) Chain rule of differentiation;

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

(3) Derivative of sigmoid function (equ. 2.44)

The in equation 2.39 required gradient of the current candidate state with respect to the previous hidden state is derived as

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

(1) Mapping 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 two vectors (equ. 2.46)

The in equation 2.41 required gradient of the current reset gate with respect to the previous hidden state shares the same structure with the gradient of equation 2.40 and is

hence derived as

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

The above derivations revert to the following equations. Equations (4.5.73) and (4.5.17) of Abramowitz and Stegun [50] 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., [39]) is given by

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

The hadamard product of two vectors can be reformulated as the matrix product of a diagonal matrix and a vector [6]

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

Together with the product rule of differentiation, the derivative of the Hadamard product of two vectors is therewith identified as

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

The derivative of a function $f : \mathbb{R} \to \mathbb{R}$, that is applied element-wise on 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 above statement can be derived from the calculation of the differential of the function

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

(1) Relation of differential and derivative

(2) Chain rule element-wise applied

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

Chapter 3

Autonomous Navigation Method

DELETEME: In this chapter you start addressing your actual problem. Therefore, it makes often sense to make a detailed problem analysis first (if not done in introduction). You should be sure about what to do and how. As writtin in the background part, it might also make sense to include complex background information or papers you are basing on in this analysis. If you are solving a software problem, you should follow the state of the art of software development which basically includes: problem analysis, design, implementation, testing, and deployment. Maintenance is often also described but I believe this will not be required for most theses. Code should be placed in the appendix unless it is solving an essential aspect of your work.

In order to investigate the effect of temporal comprehension in machine-learning-based, autonomous navigation of drones, this master thesis extends the navigation method of Kaufmann, Loquercio et. al [?] is minimally adjusted and extended with a recurrent convolutional neural network. This chapter ...

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 refer-

ence 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 figure 3.1a) is fixed to 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 figure 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.

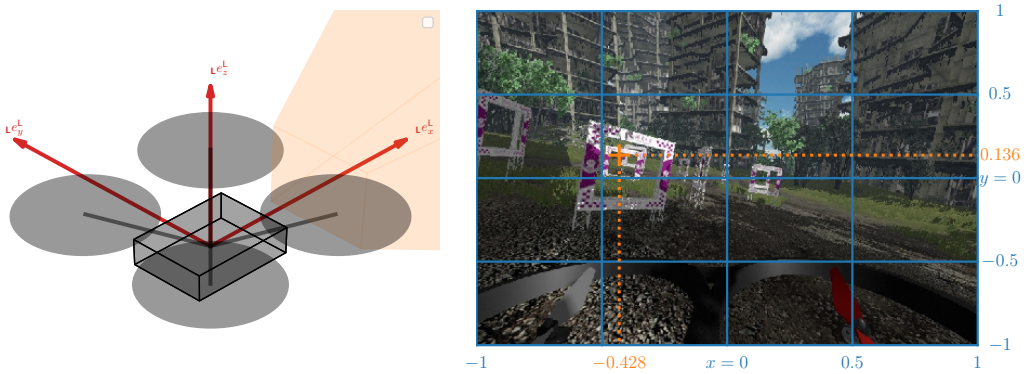
Transformation between the global and the local reference system

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., [44]. 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

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

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

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



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

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

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

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

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.

Transformation between the local and the image reference system
The horizontal $\breve{\phi}_{\text{h}}^{\text{cam}}$ and the vertical $\breve{\phi}_{\text{v}}^{\text{cam}}$ 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

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

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 devided into three steps. First, the vector from the optical center of the camera to the point to be transformed is mapped to its yaw $\operatorname{atan2}\left({}_{\mathbf{L}}y, {}_{\mathbf{L}}x\right)$ and pitch $\operatorname{atan2}\left({}_{\mathbf{L}}z, \|{}_{\mathbf{L}}\underline{p}\|_2\right)$ angle, both, with respect to the image reference system. Second these angles are normalized by the half of the horizontal $\breve{\phi}_{\text{h}}^{\text{cam}}$ and the half of the vertical $\breve{\phi}_{\text{v}}^{\text{cam}}$ 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 artifical 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

$$T_{\mathbf{LI}} : \mathbb{R}_{\geq 0}, [\text{-}1, 1]^2 \to \mathbb{R}^3; \quad d, {}_{\mathbf{I}}\underline{p} \mapsto {}_{\mathbf{L}}\underline{p} = d \begin{bmatrix} \cos\left({}_{\mathbf{L}}\phi_y\right) \\ \cos\left({}_{\mathbf{L}}\phi_y\right) \\ \sin\left({}_{\mathbf{L}}\phi_y\right) \end{bmatrix} \odot \begin{bmatrix} \cos\left({}_{\mathbf{L}}\phi_z\right) \\ \sin\left({}_{\mathbf{L}}\phi_z\right) \\ 1 \end{bmatrix}$$
$$\text{with} \quad {}_{\mathbf{L}}\phi_z = \text{-}\frac{\breve{\phi}_{\text{h}}^{\text{cam}}}{2} \cdot {}_{\mathbf{I}}x, \quad {}_{\mathbf{L}}\phi_y = \frac{\breve{\phi}_{\text{v}}^{\text{cam}}}{2} \cdot {}_{\mathbf{I}}y. \tag{3.12}$$

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 backprojection 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 backprojection length, it is assumed that the transformations approximately invert each other

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

Transformation between the global and the image reference system
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).

The ANN module learns by imitation with dataset aggregation (see section 2.2 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

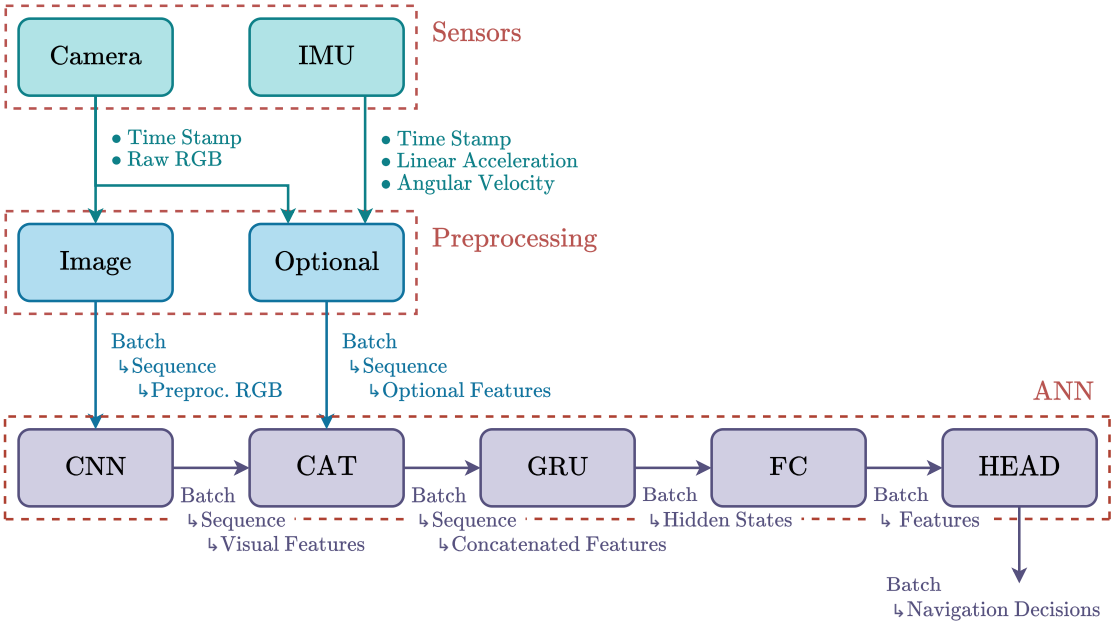


Figure 3.2: Information flow of the navigation decision making

lable. 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 racing, when the autonomous navigation method flies the drone through the racetrack, 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

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

CNN

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

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

The backbone implementation adapts to the inputted image in terms of the the number of channels $N^{\text{cnn}}_{\text{channel}}$, height $N^{\text{cnn}}_{\text{height}}$ and width $N^{\text{cnn}}_{\text{width}}$. The backbone's output dimensionality $N^{\text{cnn}}_{\text{out}}$ 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)

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

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

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

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

CAT

The CAT submodule concatenates two feature vectors

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

where the input dimensionalities $\mathbb{R}^{N^{\text{cat}}_1}$ and $\mathbb{R}^{N^{\text{cat}}_2}$ 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

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

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

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

[1] `https://pytorch.org/vision/stable/models.html`, visited on 23/08/2022

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

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

GRU

The GRU (gated recurrent unit) submodule is implemented using the PyTorch multi-layer GRU[2], which integrates the GRU [8] introduced in section 2.3 on each layer. As a quick reminder, the single layer GRU has the ability to comprehend temporal relations within sequences. Each layer $l \in \left\{1, \dots, \breve{N}_{\text{layer}}^{\text{gru}}\right\}$ processes an input sequence by iterating through it $t \in \left\{1, \dots, \breve{N}^{\text{seq}}\right\}$ 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

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

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 from the other extracted features [19]. 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

[2] `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`, visited on 24/08/2022

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 racing, 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 racing 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

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 $\odot$)

[3] `https://pytorch.org/docs/stable/nn.html`, visited on 03/07/2022

- 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 nuber of features in the given input vector. For the first layer, it adapts to the feature vector forwarded to the FC submodule $N_{\text{in},1}^{\text{fc}} = \dim(\underline{x})$. For all subsequent layers $l \geq 2$, it adapts to the width of the FC submodule $N_{\text{in},l}^{\text{fc}} = \breve{N}_{\text{width}}^{\text{fc}}$. 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

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

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

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

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

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

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

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

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

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

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

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[4], 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

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

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

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

Output

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 (see equ. 3.38) or, if it has intervened at training data generation, the expert system (see equ. **??**). 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.

[4] `https://pytorch.org/docs/stable/nn.html`, visited on 03/07/2022

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) backprojects 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 backprojection 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 devided by the slower 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

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

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[5] of the algorithm of Mueller et. al. [41] 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.

[5] `https://github.com/markwmuller/RapidQuadrocopterTrajectories`, visited on 17/08/2022

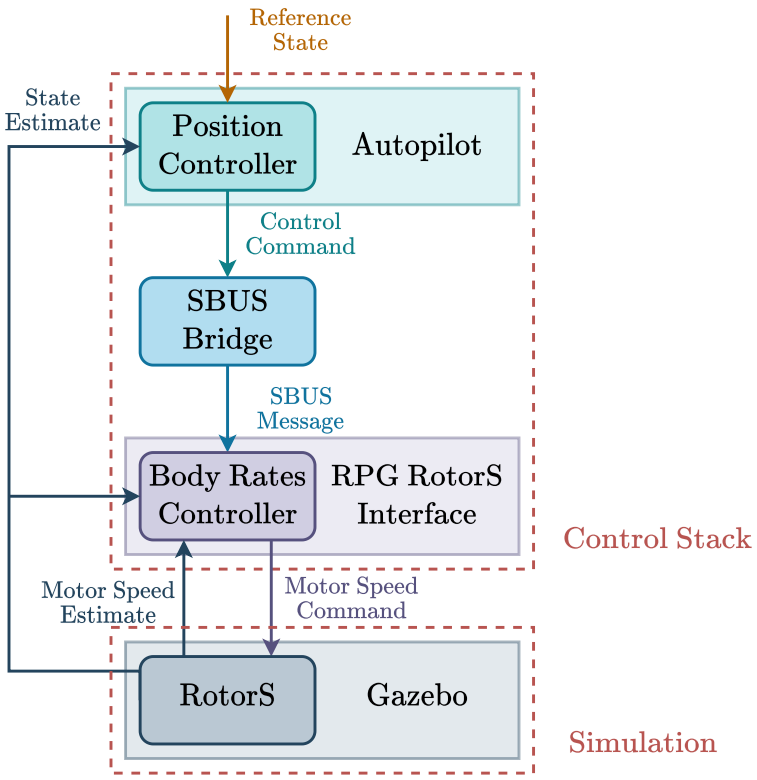


Figure 3.3: The control stack in simulation.

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 [6] implementation (see figure 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 [14]. Based on the latest reference state and the fed back drone state estimates, the position controller generates high-level control commands. A control command comprises the collective thrust of the drone's

[6] `https://github.com/uzh-rpg/rpg_quadrotor_control`, visited on 17/08/2022

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 [13]. 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. [16], is plugged into the Gazebo [7] 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 [24].

Problem

This thesis implements the expert system by Kaufmann et al. [29] in order to solve the problem of automated navigation decision making during the generation of training data for 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

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

[7] `https://gazebosim.org/home`, visited on 18/08/2022

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 [37] 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 devided by the user-specified maximum speed $\breve{v}^{\text{gt}}_{\text{max}}$ of the trajectory. The above optimization problem is temporally and spatially dedimensionalized to increase numeric stability and reformulated as quadratic program, which is solved with the Gurobi[8] 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 redimensionalized 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

[8] `https://www.gurobi.com/`, visited on 20/08/2022

user-specified margin scaled by the the user-specified drone's maximum speed, i.e.,

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

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

R4 Update the index $i^{\mathrm{gt}}_{\mathrm{proj}} \in \{0, ..., N^{\mathrm{gt}}-1\}$ 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

$$i^{\mathrm{gt}}_{\mathrm{prev}} = (i^{\mathrm{gt}}_{\mathrm{proj}} - 1 + N^{\mathrm{gt}}) \bmod N^{\mathrm{gt}}. \tag{3.67}$$

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

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

and the vector to the current drone position

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

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

$$i^{\mathrm{gt}}_{\mathrm{proj}} \leftarrow (i^{\mathrm{gt}}_{\mathrm{proj}} + 1) \bmod N^{\mathrm{gt}} \tag{3.71}$$

and go back to step 1.

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

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

the index $i^{\mathrm{gt}}_{\mathrm{proj}}$ 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

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

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

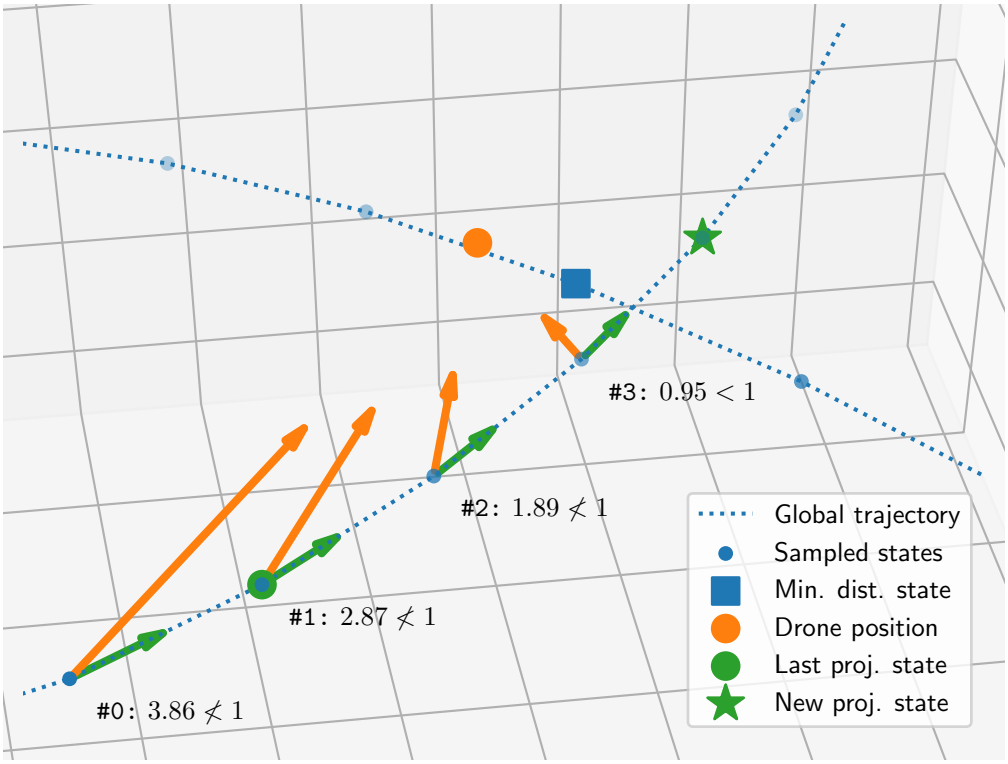


Figure 3.4: Schematic example of the update of the projection state index (**R4**). Known are: the positions (blue points) sampled from the global trajectory (blue dotted line), the last index to the projection state (green circle) and the current position of the drone (orange circle). At an iteration, 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. Figure ?? shows the related interaction of the inference engine within the autnomous navigation method. Internally, the inference engine runs the following schedule.

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 update 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.

3.6 Racing vs. Training Data Generation

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 figure 4.1) is separated into physics modeling and image rendering.

For a physics modeling of high accuracy, the Gazebo[1] simulator with the RotorS [16] 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 [55] simulator is de-

[1] `https://gazebosim.org/home`, visited on 18/08/2022

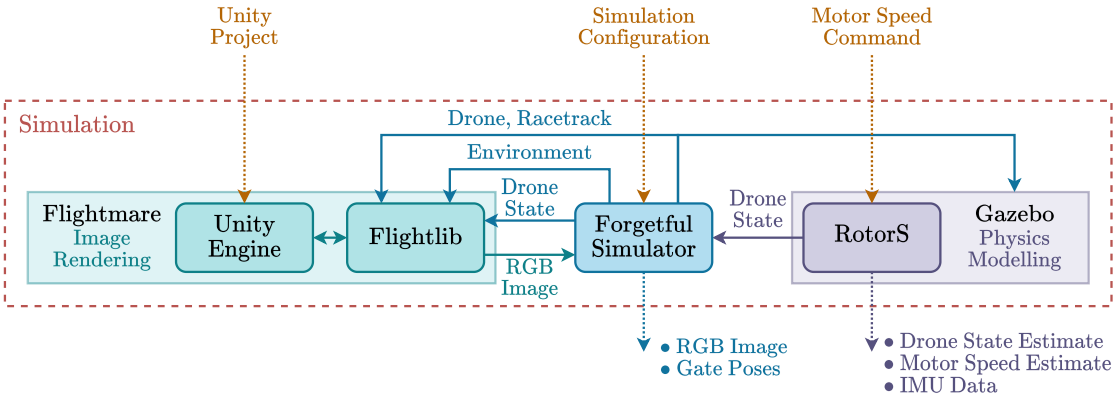
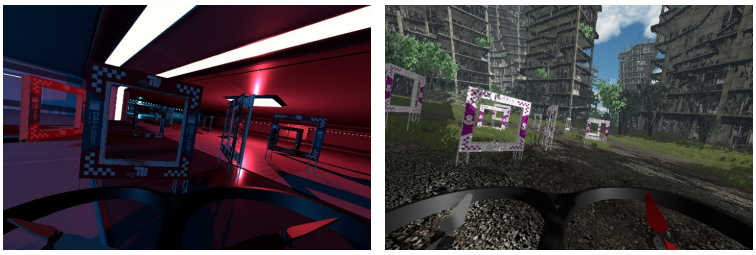


Figure 4.1: Implementation concept of the simulation



(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 that bases 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 figure 4.2). 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 figure 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 as output. Second, the node sets up the simulation according to the inputted simulation configuration.

A simulation configuration includes the environment and the racetrack configuration. The environment configuration specifies the scene and the site. The racetrack configuration specifies the racetrack type, the racetrack generation, the racetrack direc-

[2] `https://unity.com/`, visited on 20/08/2022

[3] `https://github.com/uzh-rpg/flightmare_unity`, visited on 20/08/2022

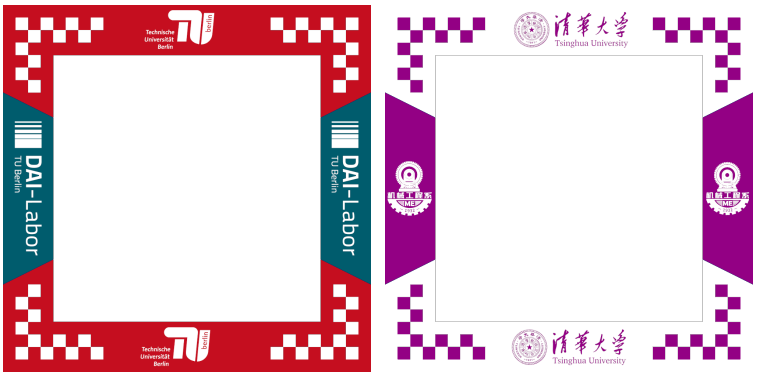
[4] based on "3D Free Modular Kit" from the Unity Asset Store

[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



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

Figure 4.3: Race gates available in simulation

tion 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: Available options for the simulation configuration

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 6.4. 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 explicitely 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 automatically update the drone's progress on the racetrack 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 of the ANN Module

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 2.2). 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 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

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 $\breve{N}^{\mathrm{seq}}$ timesteps 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 Racing Tests

After an ANN module variant completed the imitation learning process, its racing 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 racing 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 racing 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

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

In detail, the racing 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 racing 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[9] for all variants examined. A variant's number of trainable parameters determines its degree of freedom to

[9] `https://github.com/TylerYep/torchinfo`, visited on 18/08/2022

fit the aggregate training data and thus, can correlate with how well the variant performs at training and eventually at racing. 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 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 racing 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 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. [29]. The CNN submodule of F1 is, like the baseline, implemented with an 8-layer Resnet. 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 a 8-layer Resnet. Preliminary experiments on Resnet implementations of different complexity **(not documented here)** 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 racing performance of the variant.

The three recurrent variants are characterized by a deactivated FC submodule and an activated GRU submodule, which consist 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 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). **recurrent with resnet8 not converged** 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 racing 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 optional inputs available. 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 racing 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. **imagenet clasifier not converged** The pre-training of the CNN submodule was carried out with a preliminary final layer on training data from preliminary experiments **(not documented here)**. 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 racing 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[10] PyTorch implementation and update the variants' trainable parameters accordingly with the standard ADAM[11] 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)** suggest that these recurrent training epochs also lower the loss more effectively. Therefore, the number or 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 racing 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 racing performance. The starting point R1 exhibits the best racing performance among

[10] `https://pytorch.org/docs/stable/nn.html`, visited on 18/08/2022
[11] `https://pytorch.org/docs/stable/optim.html`, visited on 18/08/2022

all variants examined in experiment 1, which can be attributed to the use of the recurrent GRU submodule (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 racing 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 X samples collected from Y rollouts of the learning process. Thereafter, the variants perform racing tests with the same configuration as R1 (see table 4.4).

4.6 Design of Experiment 3

Experiment 3 studies the racing 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 equation 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 submodule, which consist 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 %. **compare with e1**

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 racing tests. The variants roll out on 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 racing performance of a recurrent variant.

The training dataset of E3R obtained in experiment 3 contains ?? samples whose input is a sequence of three pairs of an 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 module 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 with equation 4.1.

Finally, the variants perform racing tests with the same configuration as R1 (see table 4.4).

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 $men = m \times 10^n$) 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

DELETEME: The evaluation chapter is one of the most important chapters of your work. Here, you will prove usability/efficiency of your approach by presenting and interpreting your results. You should discuss your results and interprete them, if possible. Drawing conclusions on the results will be one important point that your estimators will refer to when grading your work.

In a first step, this chapter presents the results of this thesis, i.e., the performance of the examined ANN submodule variants in the racing tests (see section ...). In a second step, the results are discussed.

5.1 Results

This section presents the results from the four experiments described in chapter 4 with respect to the learning process and the racing tests of the variants examined.

Experiment 1

5.2 Discussions

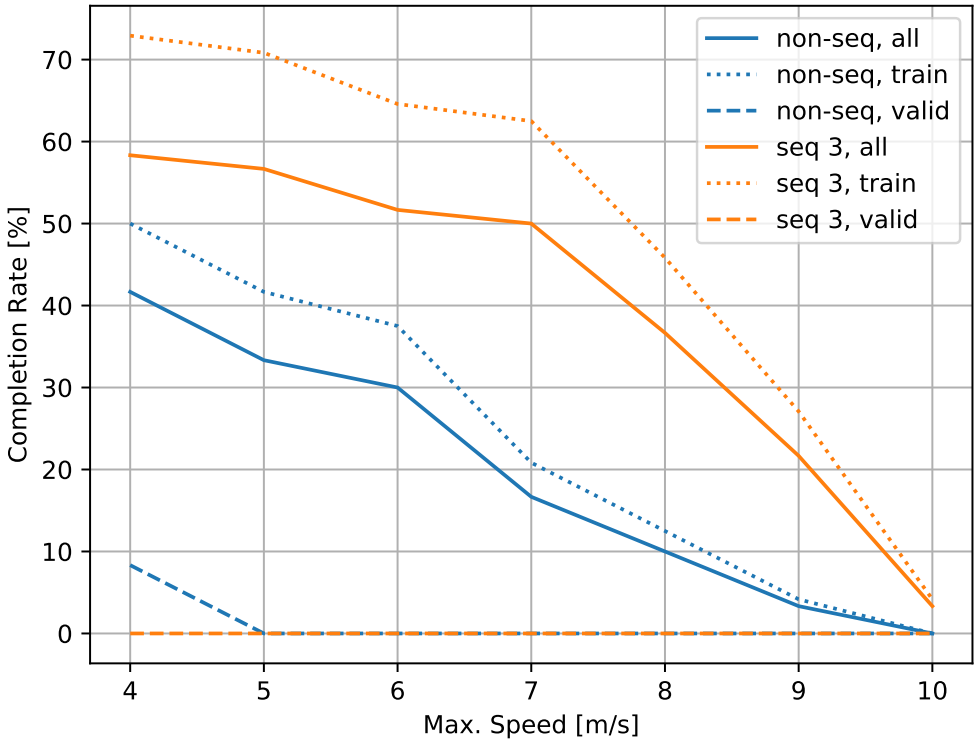

non-seq, all
non-seq, train
non-seq, valid
seq 3, all
seq 3, train
seq 3, valid
Completion Rate [%]
0
10
20
30
40
50
60
70
4
5
6
7
8
9
10
Max. Speed [m/s]

Figure 5.1: ???

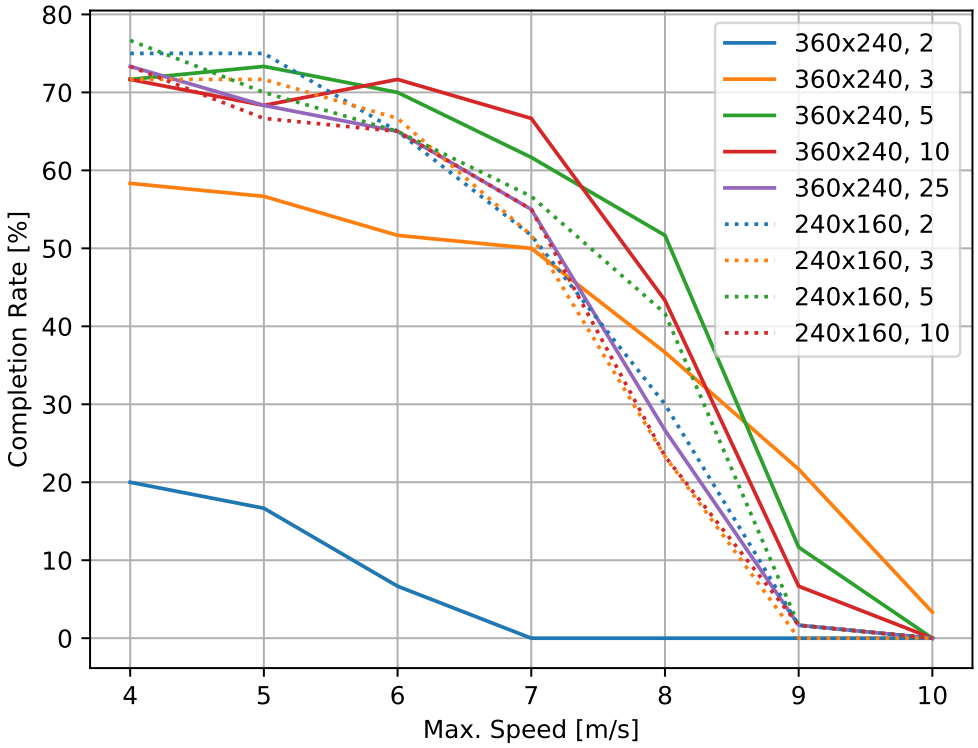

Completion Rate [%]
Max. Speed [m/s]
360x240, 2
360x240, 3
360x240, 5
360x240, 10
360x240, 25
240x160, 2
240x160, 3
240x160, 5
240x160, 10

Figure 5.2: ???

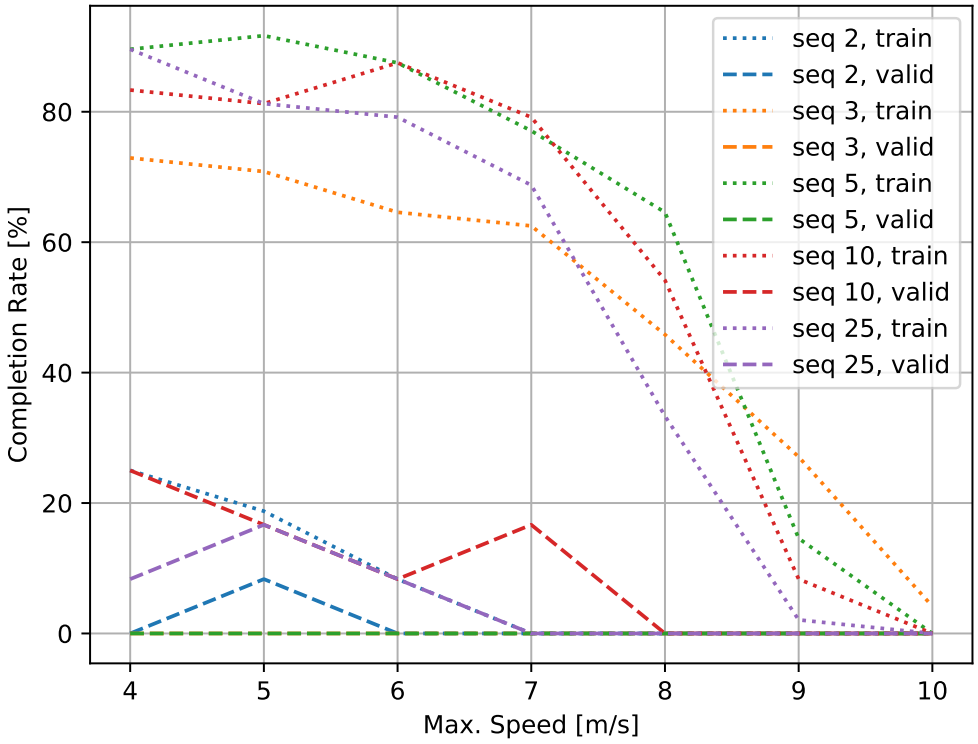

seq 2, train
seq 2, valid
seq 3, train
seq 3, valid
seq 5, train
seq 5, valid
seq 10, train
seq 10, valid
seq 25, train
seq 25, valid
Completion Rate [%]
0
20
40
60
80
4
5
6
7
8
9
10
Max. Speed [m/s]

Figure 5.3: ???

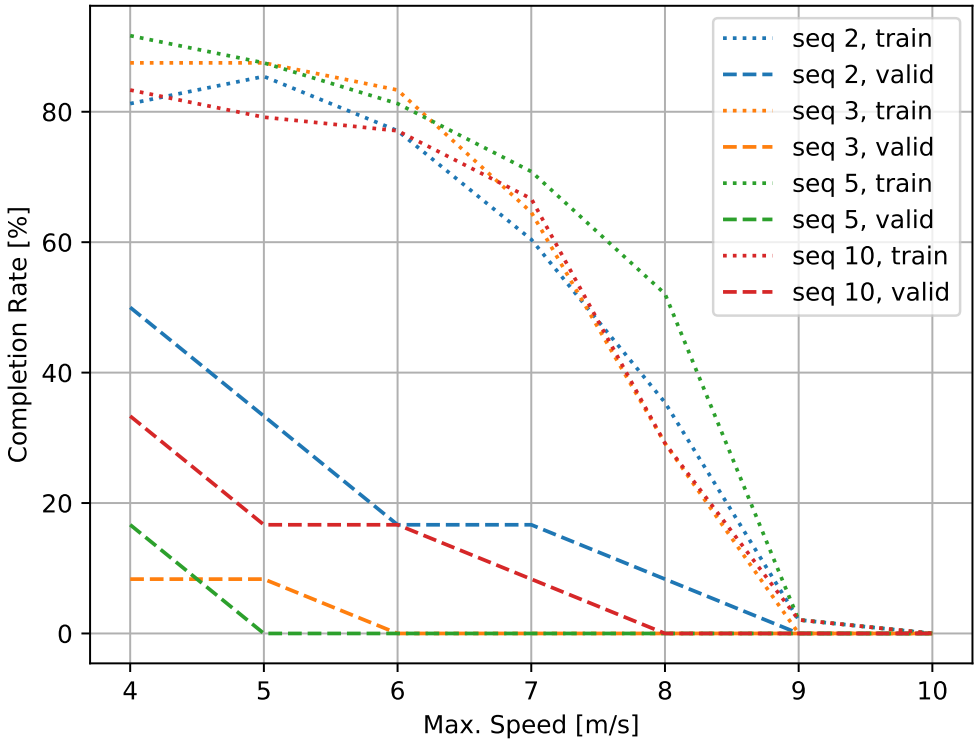

seq 2, train
seq 2, valid
seq 3, train
seq 3, valid
seq 5, train
seq 5, valid
seq 10, train
seq 10, valid
Completion Rate [%]
0
20
40
60
80
4
5
6
7
8
9
10
Max. Speed [m/s]

Figure 5.4: ???

Chapter 6

Conclusion and Future Work

6.1 Summary

DELETEME: put a plain summary of your work here. Summaries should be made of each Chapter beginning with Chapter 2 and ending with you evaluation. Just write down what you did and describe the corresponding results without reflecting on them.

6.2 Conclusion

DELETEME: do not summarize here. Reflect on the results that you have achieved. What might be the reasons and meanings of these? Did you make improvements in comparison to the state of the art? What are the good points about your results and work? What are the drawbacks?

6.3 Future Work

DELETEME: Regarding your results - which problems did you not solve? Which questions are still open? Which new questions arised? How should someone / would you continue working in your thesis field basing on your results?

- `https://en.wikipedia.org/wiki/Gated_recurrent_unit#Content-Adaptive_Recurrent_Unit`
- `https://en.wikipedia.org/wiki/Gated_recurrent_unit#Architecture`

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

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

[23] 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).

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

[25] 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.

[26] 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.

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

[28] 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.

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

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

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] Micha? Mazur, Adam Wi?niewski, Jeffery McMillan, Vicki Huff, Richard Abadie, Julian Smith, Stefan Stroh, Piotr Romanowski, Adam Kraso?, Filip Orli?ski, Agnieszka Babicz, Paulina Lulkowska, Ewa Boguszewska, Bartosz Czerkies, Adam G??b, Rozalia Ho?oga, Jakub Jagie??o, Weronika Jankowska-Tofani, Arkadiusz Kramza, Sandra Kuprijaniuk, Remigiusz Piwowarski, Jacek Sygutowski, Grzegorz Urban, Marek Walczak, Marek Wolski, and Ewa Zdrojowy. *Clarity from above*. PwC Poland, May 2016.

[37] 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.

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.

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

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

[45] 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.

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

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

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

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

[50] 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.

[51] 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.

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

[53] 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.

[54] 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).

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

[56] 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.

[57] 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.

[58] 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.

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

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

[61] 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.

[62] 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.

[63] 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

DELETEME: everything that does not fit into your work, e.g. a 5 page table that breaks the reading flow, should be placed here

Appendix A: Abbreviations

AES	Advanced Encryption Standard (Symmetrisches Verschlï¿½sselungsverfahren)
ASCII	American Standard Code for Information Interchange (Computer-Textstandard)
dpi	dots per intch (Punkte pro Zoll; Maï¿½ fï¿½r Auflï¿½sung von Bilddateien)
HTML	Hypertext Markup Language (Textbasierte Webbeschreibungssprache)
JAP	Java Anon Proxy
JPEG	Joint Photographic Experts Group (Grafikformat)
JPG	Joint Photographic Experts Group (Grafikformat; Kurzform)
LED	Light Emitting Diode (lichtemittierende Diode)
LSB	Least Significant Bit
MD5	Message Digest (Kryptographisches Fingerabdruckverfahren)
MPEG	Moving Picture Experts Group (Video- einschlieï¿½lich Audiokompression)
MP3	MPEG-1 Audio Layer 3 (Audiokompressionformat)
PACS	Picture Archiving and Communication Systems
PNG	Portable Network Graphics (Grafikformat)
RSA	Rivest, Shamir, Adleman (asymmetrisches Verschlï¿½sselungsverfahren)
SHA1	Security Hash Algorithm (Kryptographisches Fingerabdruckverfahren)
WAV	Waveform Audio Format (Audiokompressionsformat von Microsoft)

Appendix B: LaTeX Help

How to Use This Template

- Remove all of my text which is mostly labeled with DELETEME
- Change the information in the 00a_title_page.tex file
- Use the information written in this section
- Ask you supervizor to help you
- If I am not your supervizor and noone else can help you, write me an email (aubrey.schmidt@dai-labor.de)

Citations

Citing is one of the essential points you need to do in you thesis. Statements not basing on results of your own research[1] not being cited represent a breach on the rules of scientific working. Therefore, you every statement needs to be cited basing on information that other people can cross-check. A common way of citing in technical papers is:

- Oberheide et al. [**?**] state that the average time for an anti-virus enginge to be updated with a signature to detect an unknown threat is 48 days.

Note: et al. is used when the paper was written by more than two people. Check the code of this section to learn how to cite from a technical perspective.

Note: you can change the citation style in the `thesis.tex` file, e.g. to harvard style citations. Instructions on this can also be found in this file.

You should not cite anything that can be changed, e.g. it is not that good citing web pages since they might get updated changing the cited content. There are no clear quality measures on citing sources but aubrey believes that the following list is true for several cases, starting with highest quality:

1. Journal article or book
2. Conference paper
3. Workshop paper
4. Technical report
5. Master thesis

[1] in what ever context

6. Bachelor thesis

7. General Web reference

There might be workshop papers that have a higher quality than some journal papers. Therefore this list only gives you a hint on possible quality measures. Another measure can be whether a paper was indexed by ACM/IEEE, although this is not a strong indicator.

Finding and Handling Citation Sources

Following ressources are required for finding and handling articles, books, papers and sources.

- your primary resource will be `http://scholar.google.com`
- `http://www.google.com` might also be used
- `wikipedia.com` can be a good start for finding relevant papers on your topic
- you should download and install JabRef or a similar tool `http://jabref.sourceforge.net/`
- you should point JabRef to the mybib.bib file
- you should immediately enter a relevant paper to JabRef, additionally, you should write a short summary on it; else, you will do this work at least twice.

General Advices

- Do not take care of design, LaTeXwill do this for you. If you still feel that you need to take care of this, do this when you have finished writing, else you will end up in a lot of double and triple work.
- LaTeXwill do exactly that you will tell it to do. If you have problems with this, go for google or ask you supervizor
- use labels in order to be able to reference to chapters, section, subsections, figures, tables, etc. ...

General Commands

- check http://en.wikibooks.org/wiki/LaTeX
- check http://www.uni-giessen.de/hrz/tex/cookbook/cookbook.html German

Please also check the following source [**?**].

Code

This section shows you how to get your code into a LaTeXdocument. See code for options.

```
1 class Beispiel{
2
3   public static void main(String args[]){
4
5     System.out.println("Hello World");
6
7   }
8
9 }
```

```
1 class Beispiel{
2
3   public static void main(String args[]){
4
5     System.out.println("Hello World");
6
7   }
8
9 }
```

Listing 6.1: Example code is presented here

Figures

This section describes how to include images to your document. Information was taken from `http://en.wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions`, visited on 05/08/2011. Please make sure to use original vector graphics as basis since image quality might be used as weak indicator for thesis quality. For this, try to find find files in `.SVG` or `.PDF` format. Exporting a `.PNG` or `.JPG` to `.PDF` will not work since data was already lost while exporting it to these formats. This is the case for most Web graphics. Wikipedia startet entering most in images in `.SVG` which easily can be transformed to `.PDF`, but please do not forget proper citations.

Figure 6.1: Including an image; in this case a PDF. Please note that the caption is placed below the image.

Figure 6.2: See code for caption options: this is a long caption which is printed in the Text. Additionally, image size was increased

(a) Small (b) Large (c) Medium

Figure 6.3: Placing images side by side using the subfig package. Space between the images can be adjusted.

Tables

Here, you will find some example tables.The tables were taken from `http://en.wikibooks.org/wiki/LaTeX/Tables`, visited on 05/08/2011. Table environment was added plus caption and label. For code, check `__help/latex_hinweise.tex`.

Table 6.1: Simple table using vertical lines. Note that the caption is always above the table! Please check code for finding the right place for the table label.

1	2	3
4	5	6
7	8	9

Table 6.2: Table using vertical and horizontal lines

7C0	hexadecimal
3700	octal
11111000000	binary
1984	decimal

Table 6.3: Table with column width specification on last column

Day	Min Temp	Max Temp	Summary
Monday	11C	22C	A clear day with lots of sunshine. However, the strong breeze will bring down the temperatures.
Tuesday	9C	19C	Cloudy with rain, across many northern regions. Clear spells across most of Scotland and Northern Ireland, but rain reaching the far northwest.

Table 6.4: Table using multi-column and multirow

Team sheet		
Goalkeeper	GK	Paul Robinson
Defenders	LB	Lucus Radebe
	DC	Michael Duberry
	DC	Dominic Matteo
	RB	Didier Domi
Midfielders	MC	David Batty
	MC	Eirik Bakke
	MC	Jody Morris
Forward	FW	Jamie McMaster
Strikers	ST	Alan Smith
	ST	Mark Viduka

Appendix B: Racetrack Randomization and Redirection

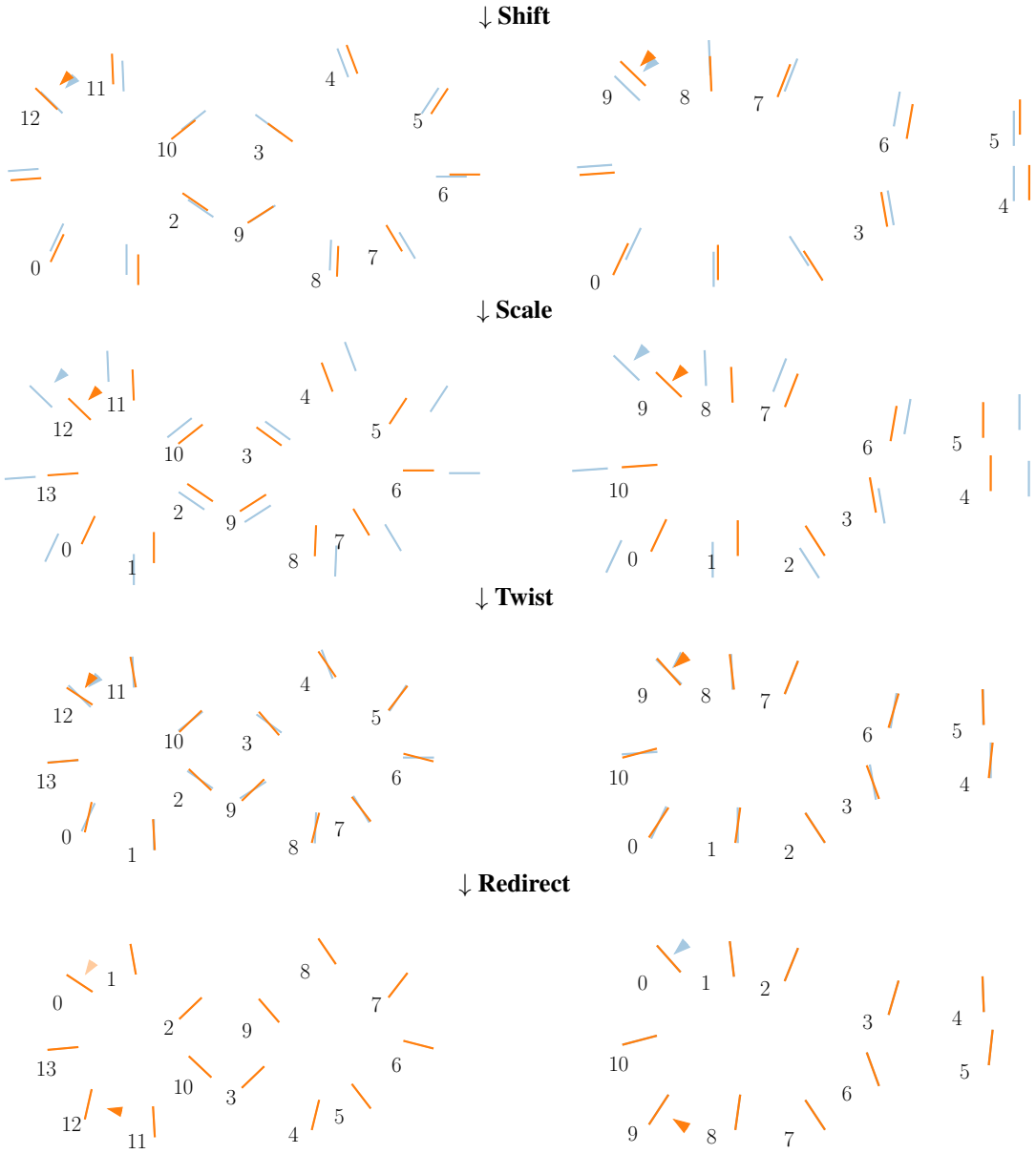


Figure 6.4: Racetrack randomization and redirection for the figure-8 (left) and the gap (right) racetrack types. In the xy-plane, the start pose of the drone (arrow) and the gate poses (numbered bars) are shown before (low-opacity blue) and after (orange) the randomizing processing steps Shift, Scale and Twist and the Redirect processing step.