Technische Universität Berlin

DOCUMENT BUILD DATE: August 31, 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

DELETEME: for readability purpose, it makes sense to write a short paragraph on what the reader can expect in this chapter.

DELETEME: tipp: sometimes it makes sense to write the first chapter, the last chapter, and the abstracts at the end. In this case, it might be easier to argue towards your topic

1.1 Motivation

DELETEME: This section is very important since it argues why it is necessary to take care of the problem you are addressing in your work. One way to do this is coming from a very broad view on the problem to a very detailled one. This can be done by establishing a chain of statements that refer to each other until you reach your particular problem. Doing this, you really need to take care for citing every statement.

DELETEME: Example for a chain: Mobile communication gets increasingly popular in the world (CITE sales on mobile communication infrastruce, mobile phones, or increasing number of mobile phones contracts). $\rightarrow$ Especially smartphones, which represent the next generation cellular phone (CITE), get more and more used for communicating not only with other people but also for connecting to the Internet for using various services (CITE). $\rightarrow$ Smartphone are comprehensive cellular phones that provide additional functionality due to their increased connection and processing capabilities (CITE). $\rightarrow$ Most smartphones offer an online application store for adding software to the devices which helps the users to customize their devices according to their needs, e.g. Android Market[1]. $\rightarrow$ One problem about installing third-party software is that not all softwares try to help the user; $\rightarrow$ software with malicious intentions, so called malicious software (malware), can be a severe threat to smarpthone users. Some malwares delete files (EXAMPLE + CITE or footnote with URL) or even destroy devices

[1]`http://market.android.com`, visited on 05/08/2011

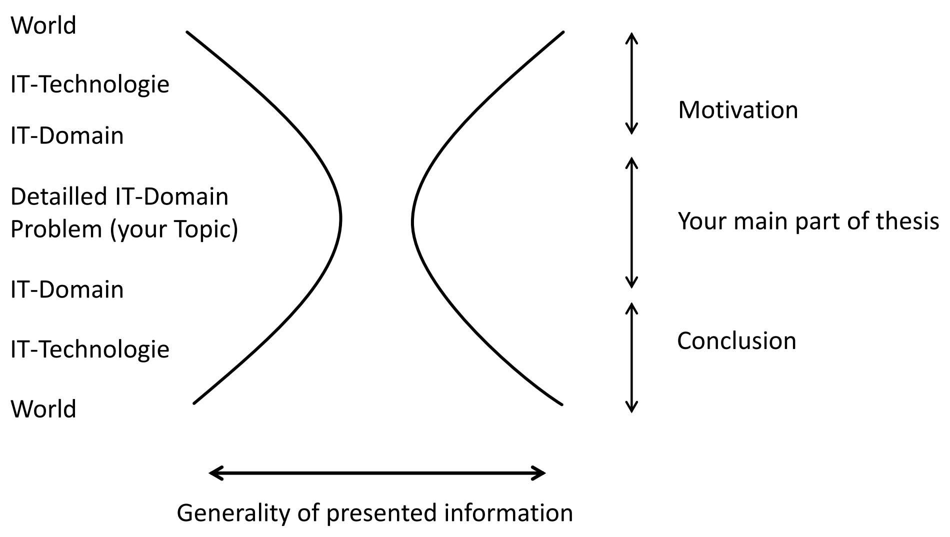


Figure 1.1: This images illustrates how generality of information could be handled in a thesis. In your motivation you should start from a very broad view on the topic. Then you should get more precise with every statement until you reach the actual problem you are addressing. You should do vice-versa in your conclusion, starting with the problem that you addressed and getting broader until you can write about the meaning of your results to the (IT-)world.

(EXAMPLE + CITE or footnote with URL). → More and more smartphone malwares appeared in the last years (CITE). → Signature-based approaches work efficiently on known malware (CITE) but face serious drawbacks regarding unknown malware. → Oberheide et al. [**?**] state that virus engines need an average time of 48 days until their databases get updated to be able to detect a certain unknown malware. → This in turn means that smartphone users stay unprotected for this time which can be seen as a severe threat. → Therefore, approaches are needed that are capable of detecting unknown malware for protecting the users against such threats. DELETEME: This example showed how one could argue that alternative approaches for malware detection is required. The length of the motivation depends on the topics handled and can of course be longer. The principle I am describing is also shown on Figure 1.1

1.2 Approach and Goals

DELETEME: In this section, you should cleary describe your approach that you are following in order to solve the underlaying problem of your thesis. Additionally, you should clearly state the goals of your work. This will not only help you supervizor to understand what you are doing, it will also help you to be sure on which topic you should evaluate.

1.3 Structure of the Thesis

DELETEME: This section does not require eloquent writing. It is just a presentation of what you will handle in each chapter starting with Chapter 2.

DELETEME: Example: This thesis is structured as follows. In Chapter 2, we discuss essential background related to the thesis topic. (SOME MORE SENTENCES). Chapter 3 represents a detailled analysis of the problem that will be addressed. In particular, (SOME MORE SENTENCES). In Chapter 4, our solution is presented. This solution covers ... (SOME MORE SENTENCES). Chapter 5 evaluates our solution basing on our specified goals. (SOME MORE SENTENCES). In Chapter 6, we conclude. 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 Direct Policy Imitation Learning

In this tutorial, we aim to present to researchers and industry practitioners a broad overview of imitation learning techniques and recent applications. Imitation learning is a powerful and practical alternative to reinforcement learning for learning sequential decision-making policies. Also known as learning from demonstrations or apprenticeship learning, imitation learning has benefited from recent progress in core learning techniques, increased availability and fidelity of demonstration data, as well as the computational advancements brought on by deep learning. We expect this tutorial to be highly relevant for researchers and practitioners who have interests in reinforcement learning, structured prediction, planning and control. The ideal audience member should have familiarity with basic supervised learning concepts. No knowledge of reinforcement learning techniques will be assumed.

Imitation learning is an area of machine learning that relates to the problem of learning to imitate (i.e., behave like) an expert. Depending on the reference, imitation learning is categorized as an ”approach” or ”alternative” to RL (reinforcement learning), or the ”fusion” of reinforcement learning and supervised learning. With respect to the degree of supervision, imitation learning relates to the three major categories of machine learning as follows. Supervised learning defines goals explicitly and thus has the highest degree of supervision. Imitation learning weakens the degree of supervision, by not defining goals, but demonstrating how to achieve them. Reinforcement learning further weakens the degree by defining rewards, replacing the concept of goals. Unsupervised learning has the lowest level because it defines neither goals nor rewards.

An imitation learning problem includes: a system, an expert, a policy class, a loss function and a learning algorithm.

The system interacts with its real or simulated environment by taking a state $\underline{x} \in \mathcal{X}$ and executing an action $\underline{u} \in \mathcal{U}$ where $\mathcal{X}$ and $\mathcal{U}$ are the domains of all possible states and actions, respectively. Under the assumption that the dynamics of the system in its environment are a MDP (Markov decision process), a probabilistic transition model can describe the system dynamics. Such a model provides the conditional probability distribution over the system state given the system state and action at the previous time step

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

The system dynamics are typically unknown to the policy to be found.

The expert (also referred to as demonstrator) is a human or system. The expert’s behaviour, i.e, the mapping from the state to the action of the same time step, is characterized by the expert policy

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

Applying the expert policy generates demonstrations. A demonstration is a state-action pair $(\underline{x}_t, \underline{u}_t)$. All demonstrations constitute a training dataset, on which the imitating policy can be trained. In this sense, ?? introduced imitation learning as a variant of supervised learning, which does not predict single iid samples at a time (assumption ...) but instead learns to predict a sequence ("learning sequential decision-making policies").

The policy class specifies the possible policies, which map from state to action. Usually, the class contains parameterized ANNs

$$\pi_{\underline{\theta}} : \mathcal{X} \to \mathcal{U}; \quad \underline{x}_t \mapsto \underline{u}_t \tag{2.3}$$

where $\underline{\theta}$ is the parameter vector of the ANNs. The policy class can be rolled out, i.e., to repeatedly apply the policy on an inital state. Thereby, it generates the trajectory

$$\tau = \{(\underline{x}_t, \pi(\underline{x}_t))\}_{t \in \{0,1,\dots\}}. \tag{2.4}$$

The imitation loss function quantifies the deviation between the decisions (trajectories) made by the imitating policy in the environment from the demonstrations of the expert.

The learning algorithm to optimize the model within the model class with respect to the imitation loss and the training data.

The target is to find a policy that, within a time step, maps states to action

$$\underline{u}_t = \pi(\underline{x}_t). \tag{2.5}$$

To do this, a set of demonstrations/trajectories is given. Each demonstration is a sequence of state-action pairs

$$\xi = \{(\underline{x}_t, \underline{u}_t)\}_{t \in \{0,1,\dots\}}. \tag{2.6}$$

generate at the roll out of the expert policy π^*.

The methods to solve an imitation learning problem can be devided into direct policy learning and inverse reinforcement learning. Direct policy learning targets at imitating the expert by learning its policy. Inverse reinforcement learning targets at learning the reward function that implicetly underlies the expert policy. As the experiments of this thesis apply direct policy learning (see section ...), this form of imitation learning is introduced in this section.

———-

First, the expert policy (also referred to as demonstrator) is a human or system that, while acting in the environment, produces state-action pairs (also referred to as demonstrations)

$$(s, a)\,. \tag{2.7}$$

The environment, which is either real or simulated, determines the state dynamics

$$P(s_{t+1}|s_t, a_t). \tag{2.8}$$

The model class of ANNs.

The training dataset is the collection of demonstrations.

Applications

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. [3], 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 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 [14] (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 [15]. RNNs train on sequential data with backpropagation through time (BPTT) [25] which is basically the application of standard backpropagation (e.g., [26]) 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

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

RNN becomes. The training of RNNs on long sequences is hence prone to the vanishing gradient problem [12], 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 [1].

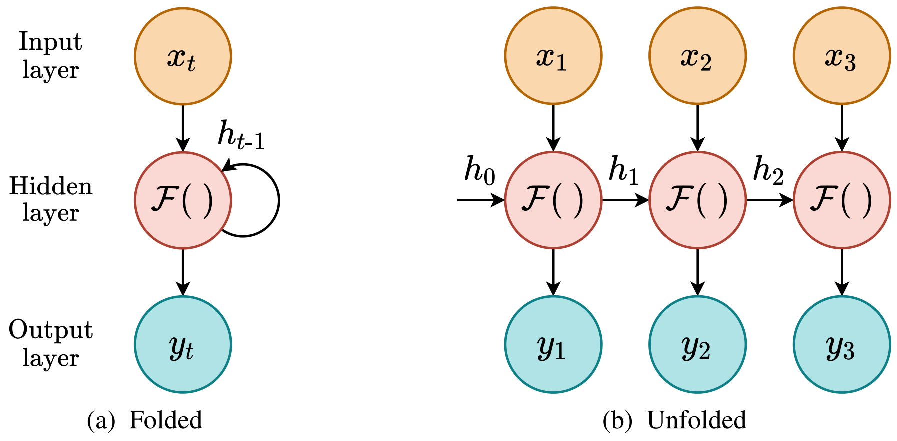


Figure 2.1: Folded schematic RNN with a single hidden layer that feeds the previous h_{t-1} (or initial h_0) state back to the inference at the current time step t and its time-unfolded representation when processing the input of a time-series consisting of the three data points $(x_i)_{i\in\{1,2,3\}}$.

In 1997, Hochreiter and Schmidhuber [13] introduced the long short-term memory (LSTM), which is today's predominant [28] 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 [25]. As a result, the LSTM is, compared to standard RNNs, essentially more capable of learning

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

to remember long-term dependencies within input sequences.

The GRU, which was proposed 17 years after the LSTM by Cho et al. [3], 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 [15]. Various empirical studies show that the GRU, compared to the LSTM, performs equally well or even slightly better. Greff et al. [9] 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. [4] 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. [30], the GRU slightly outperforms the LSTM on five of seven natural language processing tasks. In the field of algorithm learning, Kaiser and Sutskever [17] 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 accordance 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

$$\left(\underline{x}_t \in \mathbb{R}^{N^{\text{in}}}\right)_{t\in\{1,\dots,N^{\text{seq}}\},i}, \quad i \in \left\{1,\dots,N^{\text{batch}}\right\} \tag{2.9}$$

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

$$\underline{h}_{0,i} \in [-1,1]^{N^{\text{hidden}}}, \quad i \in \left\{1,\dots,N^{\text{batch}}\right\} \tag{2.10}$$

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 [31] but can also be learned by the network, e.g., [7]. The GRU layer processes the time series included in the batch parallelly and the data points of the individual time

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

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

$$\left(\underline{x}_{t,i},\ \underline{h}_{t-1,i}\right), \quad i \in \left\{1, \ldots, N^{\text{batch}}\right\}. \tag{2.11}$$

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

$$\underline{g}^{\text{r}}_{t,i} = \mathcal{F}^{\text{r}}\left(\underline{x}_{t,i}, \underline{h}_{t-1,i}\right), \quad i \in \left\{1, \ldots, N^{\text{batch}}\right\} \tag{2.12}$$

is obtained by the mapping

$$\begin{aligned} \mathcal{F}^{\text{r}} : \left(\mathbb{R}^{N^{\text{in}}},\ [-1,1]^{N^{\text{hidden}}}\right) &\to [0,1]^{N^{\text{hidden}}} \\ (\underline{x}, \underline{h}) &\mapsto \overset{\odot}{\sigma}\left(\underline{\underline{A}}^{\text{r}}_{x}\underline{x} + \underline{b}^{\text{r}}_{x} + \underline{\underline{A}}^{\text{r}}_{h}\underline{h} + \underline{b}^{\text{r}}_{h}\right). \end{aligned} \tag{2.13}$$

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

$$\begin{aligned} &\underline{\underline{A}}^{\text{r}}_{x} \in \mathbb{R}^{N^{\text{hidden}} \times N^{\text{in}}}, & &\underline{b}^{\text{r}}_{x} \in \mathbb{R}^{N^{\text{hidden}}}, \\ &\underline{\underline{A}}^{\text{r}}_{h} \in \mathbb{R}^{N^{\text{hidden}} \times N^{\text{hidden}}}, & &\underline{b}^{\text{r}}_{h} \in \mathbb{R}^{N^{\text{hidden}}}. \end{aligned} \tag{2.14}$$

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 [10] (see figure 2.2)

$$\sigma :\ \mathbb{R} \to [0,1]\,;\ x \mapsto \frac{1}{1 + e^{-x}} \tag{2.15}$$

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

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

is obtained with the mapping

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

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

$$\begin{aligned}\underline{\underline{A}}^{\mathrm{u}}_{x} &\in \mathbb{R}^{N^{\mathrm{hidden}}\times N^{\mathrm{in}}}, & \underline{b}^{\mathrm{u}}_{x} &\in \mathbb{R}^{N^{\mathrm{hidden}}},\\ \underline{\underline{A}}^{\mathrm{u}}_{h} &\in \mathbb{R}^{N^{\mathrm{hidden}}\times N^{\mathrm{hidden}}}, & \underline{b}^{\mathrm{u}}_{h} &\in \mathbb{R}^{N^{\mathrm{hidden}}}. \qquad (2.18)\end{aligned}$$

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

$$\underline{h}^{\mathrm{c}}_{t,i}=\mathcal{F}^{\mathrm{c}}\left(\underline{x}_{t,i},\underline{h}_{t-1,i}\right),\quad i\in\left\{1,\ldots,N^{\mathrm{batch}}\right\}. \qquad (2.19)$$

The mapping to the candidate state

$$\begin{aligned}\mathcal{F}^{\mathrm{c}}:\left(\mathbb{R}^{N^{\mathrm{in}}},[-1,1]^{N^{\mathrm{hidden}}}\right) &\to [-1,1]^{N^{\mathrm{hidden}}}\\ (\underline{x},\underline{h}) &\mapsto \overset{\odot}{\tanh}\left[\underline{\underline{A}}^{\mathrm{c}}_{x}\underline{x}+\underline{b}^{\mathrm{c}}_{x}+\mathcal{F}^{\mathrm{r}}(\underline{x},\underline{h})\odot\left(\underline{\underline{A}}^{\mathrm{c}}_{h}\underline{h}+\underline{b}^{\mathrm{c}}_{h}\right)\right] \qquad (2.20)\end{aligned}$$

contains the trainable weight matrices and bias vectors

$$\begin{aligned}\underline{\underline{A}}^{\mathrm{c}}_{x} &\in \mathbb{R}^{N^{\mathrm{hidden}}\times N^{\mathrm{in}}}, & \underline{b}^{\mathrm{c}}_{x} &\in \mathbb{R}^{N^{\mathrm{hidden}}},\\ \underline{\underline{A}}^{\mathrm{c}}_{h} &\in \mathbb{R}^{N^{\mathrm{hidden}}\times N^{\mathrm{hidden}}}, & \underline{b}^{\mathrm{c}}_{h} &\in \mathbb{R}^{N^{\mathrm{hidden}}}. \qquad (2.21)\end{aligned}$$

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 [27] (see figure 2.2)

$$\tanh:\ \mathbb{R}\to[-1,1];\ x\mapsto\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad (2.22)$$

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.

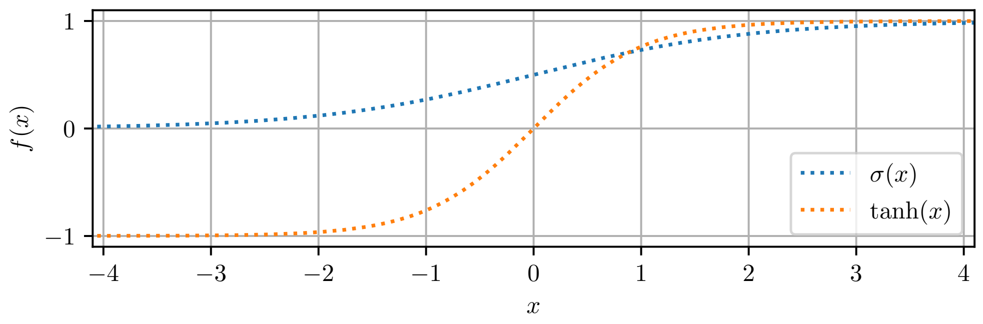


Figure 2.2: The non-linear activation functions deployed within a standard GRU. The standard sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$ normalizes the values of the reset and update gate to the interval from zero to one (equation 2.13 and 2.17). The hyperbolic tangent $\tanh(x) = (e^x - e^{-x})(e^x + e^{-x})^{-1}$ normalizes the values of the candidate and, consequently, the hidden state to the interval from minus to plus one (equation 2.20 and 2.24).

Finally, the GRU layer computes the current hidden state

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

on the basis of the mapping

$$\begin{aligned} \mathcal{F}^{\mathrm{h}} : \left(\mathbb{R}^{N^{\mathrm{in}}},\ [-1,1]^{N^{\mathrm{hidden}}}\right) &\to [-1,1]^{N^{\mathrm{hidden}}} \\ \chi := (\underline{x},\ \underline{h}) &\mapsto [\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)] \odot \mathcal{F}^{\mathrm{c}}(\chi) + \mathcal{F}^{\mathrm{u}}(\chi) \odot \underline{h}. \end{aligned} \tag{2.24}$$

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 2.20) and the values of the initial hidden state are typically initialized to be also within this interval (equation 2.10), 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 the GRU layer is aggregated from equation 2.14, 2.18 and 2.21

$$\Theta = \left\{ \underline{\underline{A}}^{\mathrm{r}}_{x}, \underline{b}^{\mathrm{r}}_{x}, \underline{\underline{A}}^{\mathrm{r}}_{h}, \underline{b}^{\mathrm{r}}_{h}, \underline{\underline{A}}^{\mathrm{u}}_{x}, \underline{b}^{\mathrm{u}}_{x}, \underline{\underline{A}}^{\mathrm{u}}_{h}, \underline{b}^{\mathrm{u}}_{h}, \underline{\underline{A}}^{\mathrm{c}}_{x}, \underline{b}^{\mathrm{c}}_{x}, \underline{\underline{A}}^{\mathrm{c}}_{h}, \underline{b}^{\mathrm{c}}_{h} \right\}. \tag{2.25}$$

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

$$N^{\text{param}} = \begin{cases} 3N^{\text{hidden}} \left(N^{\text{in}} + N^{\text{hidden}} + 2\right), & \text{if biases enabled} \\ 3N^{\text{hidden}} \left(N^{\text{in}} + N^{\text{hidden}}\right), & \text{else.} \end{cases} \tag{2.26}$$

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

$$\left(\underline{x}_t\right)_{t \in \{1,\dots,N^{\text{seq}}\},i}, \quad i \in \left\{1, \dots, N^{\text{batch}}\right\}. \tag{2.27}$$

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

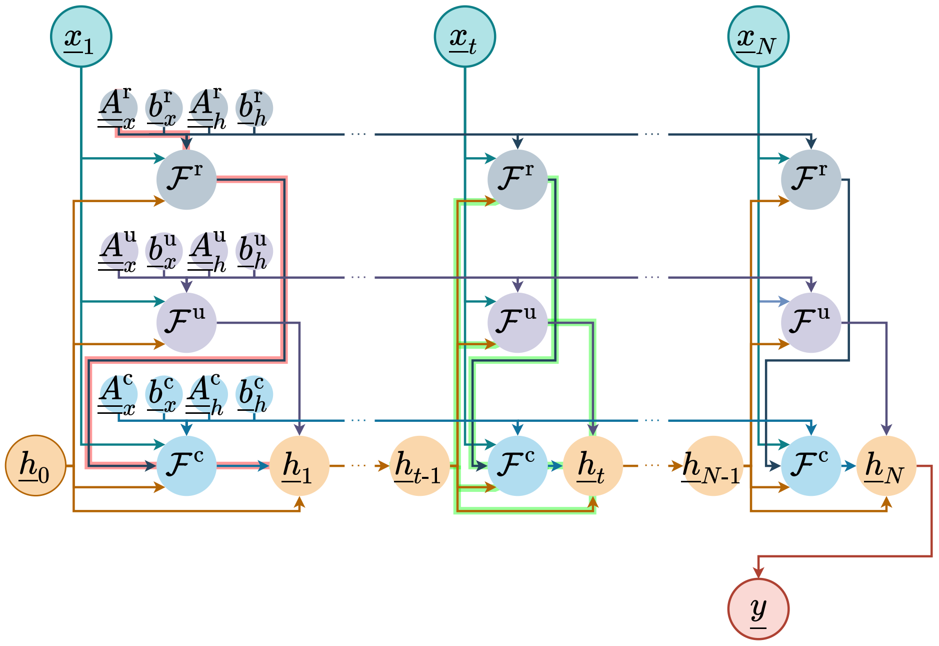
$$\underline{y}_i = \underline{h}_{N^{seq},i}, \quad i \in \left\{1, \dots, N^{\text{batch}}\right\}. \tag{2.28}$$

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 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.

$$L\left(\underline{y}_1, \dots, \underline{y}_{N^{\text{batch}}}\right) = \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} L_i\left(\underline{y}_i\right). \tag{2.29}$$

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

$$\underset{\Theta}{\operatorname{argmin}}\, L\left(\underline{y}_1, \dots, \underline{y}_{N^{\text{batch}}}\right). \tag{2.30}$$



$\underline{x}_1$
$\underline{x}_t$
$\underline{x}_N$
$\underline{\underline{A}}^{\mathrm{r}}_x$ $\underline{b}^{\mathrm{r}}_x$ $\underline{\underline{A}}^{\mathrm{r}}_h$ $\underline{b}^{\mathrm{r}}_h$
$\mathcal{F}^{\mathrm{r}}$
$\underline{\underline{A}}^{\mathrm{u}}_x$ $\underline{b}^{\mathrm{u}}_x$ $\underline{\underline{A}}^{\mathrm{u}}_h$ $\underline{b}^{\mathrm{u}}_h$
$\mathcal{F}^{\mathrm{u}}$
$\underline{\underline{A}}^{\mathrm{c}}_x$ $\underline{b}^{\mathrm{c}}_x$ $\underline{\underline{A}}^{\mathrm{c}}_h$ $\underline{b}^{\mathrm{c}}_h$
$\mathcal{F}^{\mathrm{c}}$
$\underline{h}_0$
$\underline{h}_1$
$\underline{h}_{t\text{-}1}$
$\underline{h}_t$
$\underline{h}_{N\text{-}1}$
$\underline{h}_N$
$\underline{y}$

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 2.13, 2.17 and 2.20 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 2.24 whereas the initial hidden state $\underline{h}_0$ (equation 2.10) 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 2.32) is highlighted in red for the first time step $t = 1$. The backpropagation paths of the inter-gradient (equation 2.35) is highlighted in green for the time step t.

In conformity with Li [20], the gradient of the batch loss with respect to a single element $\theta \in \Theta$ of the set of structures containing trainable parameters (equation 2.25) is computed based on backpropagation through time

$$\begin{aligned}
&\frac{\partial}{\partial\theta} L\left(\underline{y}_1, \ldots, \underline{y}_{N^{\text{batch}}}\right) \\
&\overset{(1)}{=} \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} \frac{\partial}{\partial\theta} L_i\left(\underline{y}_i\right) \\
&\overset{(2)}{=} \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} \left(\frac{\partial L_i}{\partial \underline{y}_i} \frac{\partial \underline{y}_i}{\partial\theta}\right) \\
&\overset{(3)}{=} \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} \left[\frac{\partial L_i}{\partial \underline{y}_i} \sum_{j=1}^{N^{seq}} \left(\frac{\partial \underline{y}_i}{\partial \underline{h}_{j,i}} \frac{\widehat{\partial \underline{h}_{j,i}}}{\partial\theta}\right)\right] \\
&\overset{(4)}{=} \frac{1}{N^{\text{batch}}} \sum_{i=1}^{N^{\text{batch}}} \left\{\frac{\partial L_i}{\partial \underline{y}_i} \sum_{j=1}^{N^{seq}} \left[\prod_{k=j+1}^{N^{seq}} \left(\frac{\partial \underline{h}_{k,i}}{\partial \underline{h}_{k-1,i}}\right) \frac{\widehat{\partial \underline{h}_{j,i}}}{\partial\theta}\right]\right\}.
\end{aligned} \tag{2.31}$$

(1) Loss of a batch (equ. 2.29)

(2) Chain rule

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

(4) Many-to-one output (equ. 2.28); 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 $\theta = \underline{\underline{A}}_x^{\text{r}}$ (the corresponding backpropagation path is highlighted for the first time step in figure 2.3) is examplarily derived as

$$\begin{aligned}\frac{\widehat{\partial \underline{h}_{t,i}}}{\partial \underline{\underline{A}}^{\mathrm{r}}_x} &\overset{(1)}{=} \frac{\widehat{\partial}}{\partial \underline{\underline{A}}^{\mathrm{r}}_x}\left\{\left[\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)\right] \odot \mathcal{F}^{\mathrm{c}}(\chi) + \mathcal{F}^{\mathrm{u}}(\chi) \odot \underline{h}_{t\text{-}1,i}\right\} \\ &\overset{(2)}{=} \operatorname{diag}\left[\underline{1} - \mathcal{F}^{\mathrm{u}}(\chi)\right] \frac{\widehat{\partial \mathcal{F}^{\mathrm{c}}(\chi)}}{\partial \underline{\underline{A}}^{\mathrm{r}}_x}\end{aligned} \tag{2.32}$$

(1) Mapping to hidden state (equ. 2.23, 2.24); $\chi := \left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)$

(2) Sum rule of differentiation;

Mapping to update gate (equ. 2.17): $\widehat{\partial \mathcal{F}^{\mathrm{u}} / \partial \underline{\underline{A}}^{\mathrm{r}}_x} = 0$

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. 2.41)

with

$$\begin{aligned}\frac{\widehat{\partial \mathcal{F}^{\mathrm{c}}(\chi)}}{\partial \underline{\underline{A}}^{\mathrm{r}}_x} &\overset{(1)}{=} \frac{\widehat{\partial}}{\partial \underline{\underline{A}}^{\mathrm{r}}_x}\left\{\overset{\odot}{\tanh}\left[\underbrace{\underline{\underline{A}}^{\mathrm{c}}_x \underline{x}_{t,i} + \underline{b}^{\mathrm{c}}_x + \mathcal{F}^{\mathrm{r}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right) \odot \left(\underline{\underline{A}}^{\mathrm{c}}_h \underline{h}_{t\text{-}1,i} + \underline{b}^{\mathrm{c}}_h\right)}_{:=\chi^{\mathrm{c}}}\right]\right\} \\ &\overset{(2)}{=} \operatorname{diag}\left[\frac{\partial}{\partial \chi^{\mathrm{c}}} \overset{\odot}{\tanh}\left(\chi^{\mathrm{c}}\right)\right] \cdot \frac{\widehat{\partial \chi^{\mathrm{c}}}}{\partial \underline{\underline{A}}^{\mathrm{r}}_x} \\ &\overset{(3)}{=} \operatorname{diag}\left[1 - \overset{\odot}{\tanh}{}^2\left(\chi^{\mathrm{c}}\right)\right] \cdot \operatorname{diag}\left(\underline{\underline{A}}^{\mathrm{c}}_h \underline{h}_{t-1,i} + \underline{b}^{\mathrm{c}}_h\right) \frac{\partial \mathcal{F}^{\mathrm{r}}\widehat{\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)}}{\partial \underline{\underline{A}}^{\mathrm{r}}_x}\end{aligned} \tag{2.33}$$

(1) Mapping to candidate state (equ. 2.20); $\chi := \left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)$

(2) Chain rule of differentiation;

Derivative of function applied element-wise on vector (equ. 2.43)

(3) Derivative of hyperbolic tangent (equ. 2.39);

Hadamard product of two vectors (equ. 2.41) and

$$
\begin{aligned}
\frac{\partial \mathcal{F}^{\mathrm{r}}\,\widehat{\left(\underline{x}_{t,i},\underline{h}_{t\text{-}1,i}\right)}}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}} &\overset{(1)}{=} \frac{\widehat{\partial}}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}} \left[\overset{\odot}{\sigma} \left(\underbrace{\underline{\underline{A}}^{\mathrm{r}}_{x}\underline{x}_{t,i} + \underline{b}^{\mathrm{r}}_{x} + \underline{\underline{A}}^{\mathrm{r}}_{h}\underline{h}_{t\text{-}1,i} + \underline{b}^{\mathrm{r}}_{h}}_{:=\chi^{\mathrm{r}}} \right) \right] \\
&\overset{(2)}{=} \operatorname{diag}\left[\frac{\partial}{\partial \chi^{\mathrm{r}}} \overset{\odot}{\sigma}\left(\chi^{\mathrm{r}}\right)\right] \cdot \frac{\widehat{\partial \chi^{\mathrm{u}}}}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}} \\
&\overset{(3)}{=} \operatorname{diag}\left\{\overset{\odot}{\sigma}\left(\chi^{\mathrm{r}}\right) \odot \left[1 - \overset{\odot}{\sigma}\left(\chi^{\mathrm{r}}\right)\right]\right\} \cdot \frac{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}\underline{x}_{t,i}}{\partial \underline{\underline{A}}^{\mathrm{r}}_{x}}. \qquad (2.34)
\end{aligned}
$$

(1) Mapping to reset gate (equ. 2.13)

(2) Chain rule of differentiation;

Derivative of function applied element-wise on vector (equ. 2.43)

(3) Derivative of sigmoid function (equ. 2.40)

For the computation of the gradient $\partial\left(\underline{\underline{A}}^{\mathrm{r}}_{x}\underline{x}_{t,i}\right)/\partial \underline{\underline{A}}^{\mathrm{r}}_{x}$, which is a 3-dimensional matrix whose information content is only 2-dimensional, refer to [19], for example.

The in equation 2.31 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

$$
\begin{aligned}
\frac{\partial \underline{h}_{t,i}}{\partial \underline{h}_{t\text{-}1,i}} &\overset{(1)}{=} \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \left\{\left[\underline{1} - \mathcal{F}^{\mathrm{u}}\left(\chi\right)\right] \odot \mathcal{F}^{\mathrm{c}}\left(\chi\right) + \mathcal{F}^{\mathrm{u}}\left(\chi\right) \odot \underline{h}_{t\text{-}1,i}\right\} \\
&\overset{(2)}{=} \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \left\{\left[\underline{1} - \mathcal{F}^{\mathrm{u}}\left(\chi\right)\right] \odot \mathcal{F}^{\mathrm{c}}\left(\chi\right)\right\} + \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \left\{\mathcal{F}^{\mathrm{u}}\left(\chi\right) \odot \underline{h}_{t-1,i}\right\} \\
&\overset{(3)}{=} -\operatorname{diag}\left[\mathcal{F}^{\mathrm{c}}\left(\chi\right)\right] \frac{\partial \mathcal{F}^{\mathrm{u}}\left(\chi\right)}{\partial \underline{h}_{t\text{-}1,i}} + \operatorname{diag}\left[\underline{1} - \mathcal{F}^{\mathrm{u}}\left(\chi\right)\right] \frac{\partial \mathcal{F}^{\mathrm{c}}\left(\chi\right)}{\partial \underline{h}_{t\text{-}1,i}} \\
&\qquad + \operatorname{diag}\left(\underline{h}_{t-1,i}\right) \frac{\partial \mathcal{F}^{\mathrm{u}}\left(\chi\right)}{\partial \underline{h}_{t\text{-}1,i}} + \operatorname{diag}\left[\mathcal{F}^{\mathrm{u}}\left(\chi\right)\right]. \qquad (2.35)
\end{aligned}
$$

(1) Mapping to current hidden state (equ. 2.23, 2.24); $\chi := \left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)$

(2) Sum rule of differentiation

(3) Differentiation of Hadamard product of two vectors (equ. 2.42)

The in equation 2.35 required gradient of the current update gate with respect to the previous hidden state is derived as

$$\frac{\partial \mathcal{F}^{\mathrm{u}}\left(\underline{x}_{t,i}, \underline{h}_{t\text{-}1,i}\right)}{\partial \underline{h}_{t\text{-}1,i}} \overset{(1)}{=} \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \left[\overset{\odot}{\sigma} \left(\underbrace{\underline{\underline{A}}^{\mathrm{u}}_{x} \underline{x}_{t,i} + \underline{b}^{\mathrm{u}}_{x} + \underline{\underline{A}}^{\mathrm{u}}_{h} \underline{h}_{t\text{-}1,i} + \underline{b}^{\mathrm{u}}_{h}}_{:=\chi^{\mathrm{u}}} \right) \right]$$
$$\overset{(2)}{=} \operatorname{diag}\left[\frac{\partial}{\partial \chi^{\mathrm{u}}} \overset{\odot}{\sigma}\left(\chi^{\mathrm{u}}\right) \right] \cdot \frac{\partial}{\partial \underline{h}_{t\text{-}1,i}} \chi^{\mathrm{u}}$$
$$\overset{(3)}{=} \operatorname{diag}\left\{ \overset{\odot}{\sigma}\left(\chi^{\mathrm{u}}\right) \odot \left[1 - \overset{\odot}{\sigma}\left(\chi^{\mathrm{u}}\right) \right] \right\} \cdot \underline{\underline{A}}^{\mathrm{u}}_{h}. \tag{2.36}$$

(1) Mapping to update gate (equ. 2.17)

(2) Chain rule of differentiation;

Derivative of function applied element-wise on vector (equ. 2.43)

(3) Derivative of sigmoid function (equ. 2.40)

The in equation 2.35 required gradient of the current candidate state with respect to the previous hidden state is derived as

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

(1) Mapping to candidate state (equ. 2.20)

(2) Chain rule of differentiation;

Derivative of function applied element-wise on vector (equ. 2.43)

(3) Derivative of hyperbolic tangent (equ. 2.39);

Differentiation of Hadamard product of two vectors (equ. 2.42)

The in equation 2.37 required gradient of the current reset gate with respect to the previous hidden state shares the same structure with the gradient of equation 2.36 and is hence derived as

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

$$\text{with } \chi^{\mathrm{r}} := \underline{\underline{A}}^{\mathrm{r}}_{x} \underline{x}_{t,i} + \underline{b}^{\mathrm{r}}_{x} + \underline{\underline{A}}^{\mathrm{r}}_{h} \underline{h}_{t\text{-}1,i} + \underline{b}^{\mathrm{r}}_{h}. \tag{2.38}$$

The above derivations revert to the following equations. Equations (4.5.73) and (4.5.17) of Abramowitz and Stegun [27] yield the derivative of the hyperbolic tangent as

$$\frac{\mathrm{d}}{\mathrm{d}x} \tanh(x) = 1 - \tanh^2(x). \tag{2.39}$$

The derivative of the sigmoid function (e.g., [22]) is given by

$$\frac{\mathrm{d}}{\mathrm{d}x} \sigma\left(x\right) = \sigma\left(x\right)\left[1 - \sigma\left(x\right)\right]. \tag{2.40}$$

The hadamard product of two vectors can be reformulated as the matrix product of a diagonal matrix and a vector [2]

$$\underline{x}_1 \odot \underline{x}_2 = \operatorname{diag}\left(\underline{x}_1\right) \underline{x}_2 = \operatorname{diag}\left(\underline{x}_2\right) \underline{x}_1. \tag{2.41}$$

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

$$\frac{\partial}{\partial t}\left(\underline{x}_1 \odot \underline{x}_2\right) = \operatorname{diag}\left(\underline{x}_2\right) \frac{\partial \underline{x}_1}{\partial t} + \operatorname{diag}\left(\underline{x}_1\right) \frac{\partial \underline{x}_2}{\partial t}. \tag{2.42}$$

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

$$\frac{\partial}{\partial \underline{x}} \overset{\odot}{f}\left(\underline{x}\right) = \operatorname{diag}\left[\overset{\odot}{f'}\left(\underline{x}\right)\right]. \tag{2.43}$$

The above statement can be derived from the calculation of the differential of the function

$$\frac{\mathrm{d}}{\mathrm{d}\underline{x}} \overset{\odot}{f}\left(\underline{x}\right) \cdot \mathrm{d}\underline{x} \overset{(1)}{=} \mathrm{d}\overset{\odot}{f}\left(\underline{x}\right) \overset{(2)}{=} \left[\overset{\odot}{f'}\left(\underline{x}\right)\right] \odot \mathrm{d}\underline{x} \overset{(3)}{=} \operatorname{diag}\left[\overset{\odot}{f'}\left(\underline{x}\right)\right] \cdot \mathrm{d}\underline{x}. \tag{2.44}$$

(1) Relation of differential and derivative

(2) Chain rule element-wise applied

(3) Hadamard product of two vectors (equ. 2.41) (2.45)

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 reference system, equates to the standard basis of $\mathbb{R}^3$

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

The local reference system (see 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.

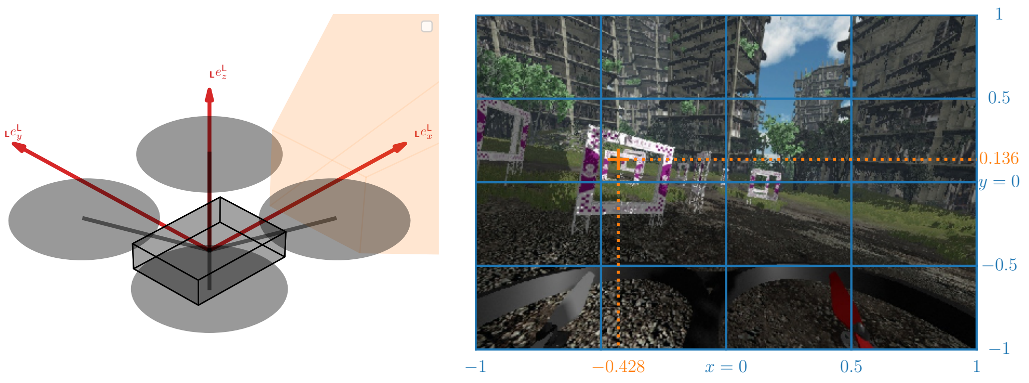
Transformation between the global and the local reference system

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., [24]. 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

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



$_{\mathbf{L}}e_z^{\mathbf{L}}$
$_{\mathbf{L}}e_y^{\mathbf{L}}$
$_{\mathbf{L}}e_x^{\mathbf{L}}$
1
0.5
0.136
$y = 0$
-0.5
-1
-1
-0.428
$x = 0$
0.5
1

(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 $_{\mathbf{I}}\underline{p}^{\mathrm{wp}} = \begin{bmatrix} -0.428 & 0.136 \end{bmatrix}^T$ (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

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

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

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

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

Finally, the inversion of a quaternion is given by

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

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

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

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

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

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

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 $\breve{\phi}_{\mathrm{h}}^{\mathrm{cam}}$ and the half of the vertical $\breve{\phi}_{\mathrm{v}}^{\mathrm{cam}}$ angle of view of the camera, respectively. Third, these normalized angles are bounded to be in the interval from minus to plus one. This boundary takes into account that an 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

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

In the above transformation, the operator $\odot$ denotes the Hadamard product, i.e., the element-wise product of two equally dimensioned matrices. Because the 2D coordinates of the image reference system can only contain information about the direction of a point, the above transformation to 3D requires the additional input of a 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 module's information flow. The 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 differ in that the user specifies the design parameters of all (inner and outer) individual submodules. This includes, among other things, the de-/activation of individual inner submodules (CAT, GRU and FC). A deactivated inner submodule recedes to the identity map. 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 is trained with supervised learning on batches whose elements are randomly sampled from training data of sequential samples. The user specifies the batch size $\breve{N}^{\text{batch}} \in \mathbb{N}_{>0}$ and the sequence length $\breve{N}^{\text{seq}} \in \mathbb{N}_{>0}$ of the training samples for the

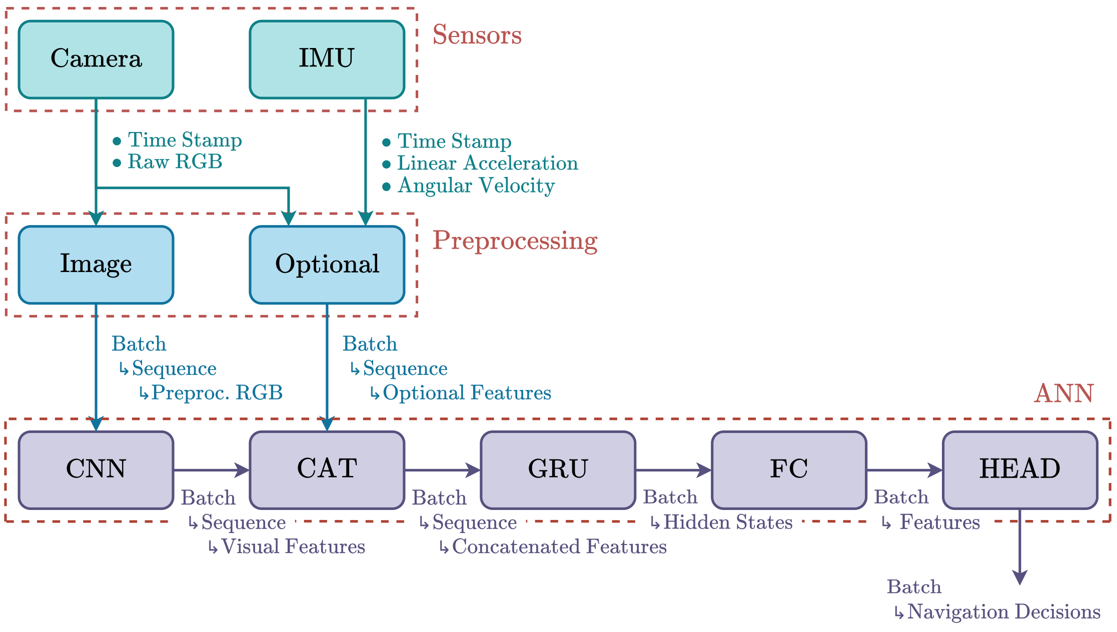


Figure 3.2: Perception and Reasoning

individual ANN module variant. The submodules prior to the GRU submodule (CNN and CAT) have no awareness of sequential connections and, thus, process sequence elements in parallel like batch elements. The GRU submodule, which has the sequential awareness, operates in many-to-one mode, i.e., 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 training data.

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

$$\underline{\underline{\underline{x}}}^{\text{cam}} \in \{0, \dots, N^{\text{cam}}_{\text{max}}\}^{3 \times N^{\text{cam}}_{\text{height}} \times N^{\text{cam}}_{\text{width}}} \tag{3.16}$$

where $N^{\text{cam}}_{\text{max}}$ (usually $= 255$) is the full pixel intensity and $N^{\text{cam}}_{\text{height}} \times N^{\text{cam}}_{\text{width}}$ 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 $N^{\text{cam}}_{\text{max}}$ 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 $\breve{s}^{\text{cnn}}_{\text{resize}}$. 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

$$\underline{\underline{\underline{x}}}^{\text{preproc}} \in \left\{ \frac{i}{N^{\text{cam}}_{\text{max}}} \right\}_{i \in \{0,\dots,N^{\text{cam}}_{\text{max}}\}}^{3 \times \left\lfloor \breve{s}^{\text{cnn}}_{\text{resize}} \cdot N^{\text{cam}}_{\text{height}} \right\rfloor \times \left\lfloor \breve{s}^{\text{cnn}}_{\text{resize}} \cdot N^{\text{cam}}_{\text{width}} \right\rfloor} \tag{3.17}$$

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

The available optional input features are

- the time step t^{cam}_{Δ} 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 ${}_{\mathbf{L}}\underline{\hat{a}}^{\text{imu}} \in \mathbb{R}^3$ and angular velocity ${}_{\mathbf{L}}\underline{\hat{\omega}}^{\text{imu}} \in \mathbb{R}^3$ in the local reference system
- the time step t^{imu}_{Δ} 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

$$\underline{x}^{\text{opt}} = \begin{bmatrix} t^{\text{cam}}_{\Delta} \\ {}_{\mathbf{L}}\underline{\hat{a}}^{\text{imu}} \\ {}_{\mathbf{L}}\underline{\hat{\omega}}^{\text{imu}} \\ t^{\text{imu}}_{\Delta} \end{bmatrix}. \tag{3.18}$$

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

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

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

$$\breve{\mathcal{F}}^{\text{cnn}} : \mathbb{R}^{N^{\text{cnn}}_{\text{channel}} \times N^{\text{cnn}}_{\text{height}} \times N^{\text{cnn}}_{\text{width}}} \to \mathbb{R}^{N^{\text{cnn}}_{\text{out}}}; \quad \underline{\underline{\underline{x}}} \mapsto \underline{x}^{\text{vis}}. \tag{3.19}$$

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. 3.17)

$$\left(\underline{\underline{\underline{x}}}^{\text{preproc}}_t\right)_{t \in \left\{1,\dots,\breve{N}^{\text{seq}}\right\},i}, \quad i \in \left\{1,\dots,\breve{N}^{\text{batch}}\right\}. \tag{3.20}$$

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

$$\left(\underline{x}^{\text{vis}}_t\right)_{t \in \left\{1,\dots,\breve{N}^{\text{seq}}\right\},i}, \quad i \in \left\{1,\dots,\breve{N}^{\text{batch}}\right\}. \tag{3.21}$$

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

$$\mathcal{F}^{\text{cat}} : \left(\mathbb{R}^{N^{\text{cat}}_1}, \mathbb{R}^{N^{\text{cat}}_2}\right) \to \mathbb{R}^{N^{\text{cat}}_1 + N^{\text{cat}}_2}; \quad (\underline{x}_1, \underline{x}_2) \mapsto \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} \tag{3.22}$$

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. 3.21) and optional input feature vector

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

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

$$\left(\begin{bmatrix} \underline{x}^{\text{vis}}_t \\ \underline{x}^{\text{opt}}_t \end{bmatrix}\right)_{t \in \left\{1,\dots,\breve{N}^{\text{seq}}\right\},i}, \quad i \in \left\{1,\dots,\breve{N}^{\text{batch}}\right\}. \tag{3.24}$$

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

$$N^{\text{cat}}_{\text{params}} = 0. \tag{3.25}$$

GRU

The GRU (gated recurrent unit) submodule is implemented using the PyTorch multi-layer GRU[2], which integrates the GRU [3] 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 $l \in \left\{1, \dots, \breve{N}^{\text{gru}}_{\text{layer}}\right\}$ 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. 2.23)

$$\mathcal{F}^{\text{gru}}_{l} : \left(\mathbb{R}^{N^{\text{gru}}_{\text{in},l}}, [-1,1]^{\breve{N}^{\text{gru}}_{\text{hidden}}}\right) \to [-1,1]^{\breve{N}^{\text{gru}}_{\text{hidden}}}; \quad \left(\underline{x}^{(l)}_{t}, \underline{h}^{(l)}_{t-1}\right) \mapsto \underline{h}^{(l)}_{t}. \tag{3.26}$$

Thereby, $\underline{h}^{(l)}_{0}$ 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 $\breve{N}^{\text{gru}}_{\text{layer}}$ and the hidden size $\breve{N}^{\text{gru}}_{\text{hidden}}$ (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

$$\begin{aligned}
\mathcal{F}^{\text{gru}} :\ & \left(\mathbb{R}^{N^{\text{gru}}_{\text{in},1}}, [-1,1]^{\breve{N}^{\text{gru}}_{\text{hidden}} \times \breve{N}^{\text{gru}}_{\text{layer}}}\right) \to [-1,1]^{\breve{N}^{\text{gru}}_{\text{hidden}}} \\
& \left(\underline{x}_{t}, \underline{h}^{(1)}_{t-1}, \dots, \underline{h}^{(\breve{N}^{\text{gru}}_{\text{layer}})}_{t-1}\right) \mapsto \underline{h}^{(\breve{N}^{\text{gru}}_{\text{layer}})}_{t} = \dots \\
& \dots \mathcal{F}^{\text{gru}}_{\breve{N}^{\text{gru}}_{\text{layer}}}\left(\underline{\delta} \odot \mathcal{F}^{\text{gru}}_{\breve{N}^{\text{gru}}_{\text{layer}}-1}\left(\underline{\delta} \odot \dots \mathcal{F}^{\text{gru}}_{1}\left(\underline{x}_{t}, \underline{h}^{(1)}_{t-1}\right), \underline{h}^{(\breve{N}^{\text{gru}}_{\text{layer}}-1)}_{t-1}\right), \underline{h}^{(\breve{N}^{\text{gru}}_{\text{layer}})}_{t-1}\right).
\end{aligned} \tag{3.27}$$

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 [11]. Mathematically, dropout applied on a hidden state (or feature vector) is calculated with the Hadamard product (denoted with $\odot$) of that hidden state and the vector $\underline{\delta}$ of same dimensionality whose entries, for every calculation, are random variables resampled from a Bernoulli distribution with a user-specified probability

$$P\left(\delta_i = 0\right) = \breve{p}^{\text{gru}}, \quad P\left(\delta_i = \frac{1}{1 - \breve{p}^{\text{gru}}}\right) = 1 - \breve{p}^{\text{gru}}. \tag{3.28}$$

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

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

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. 2.26), the total number of trainable parameters of the GRU submodule is

$$N^{\text{gru}}_{\text{params}} = 3\breve{N}^{\text{gru}}_{\text{hidden}} \Big((N^{\text{gru}}_{\text{in},1} + \breve{N}^{\text{gru}}_{\text{hidden}} + 2) + (\breve{N}^{\text{gru}}_{\text{layer}} - 1)(2\breve{N}^{\text{gru}}_{\text{hidden}} + 2) \Big) . \tag{3.29}$$

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. 3.24) to a batch of last hidden states of the last layer

$$\underline{h}^{(\breve{N}^{\text{gru}}_{\text{layer}})}_{\breve{N}^{\text{seq}},i}, \quad i \in \left\{1, \dots, \breve{N}^{\text{batch}}\right\} . \tag{3.30}$$

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 $l \in \left\{1, \dots, \breve{N}^{\text{fc}}_{\text{layer}}\right\}$ applies an activation, dropout and a biased linear transformation on the input feature vector

$$\mathcal{F}^{\text{fc}}_{l} : \mathbb{R}^{N^{\text{fc}}_{\text{in},l}} \to \mathbb{R}^{\breve{N}^{\text{fc}}_{\text{width}}}; \quad \underline{x} \mapsto \underline{\underline{A}}^{\text{fc}}_{l} \cdot \underline{\delta}^{\text{fc}} \odot \overset{\odot}{\breve{f}}{}^{\text{fc}}(\underline{x}) + \underline{b}^{\text{fc}}_{l} . \tag{3.31}$$

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

$$\mathcal{F}^{\text{fc}} : \mathbb{R}^{N^{\text{fc}}_{\text{in},1}} \to \mathbb{R}^{\breve{N}^{\text{fc}}_{\text{width}}}; \quad \underline{x} \mapsto \mathcal{F}^{\text{fc}}_{\breve{N}^{\text{fc}}_{\text{layer}}} \Big(\mathcal{F}^{\text{fc}}_{\breve{N}^{\text{fc}}_{\text{layer}}-1} \big(\dots \mathcal{F}^{\text{fc}}_{1}(\underline{x}) \big) \Big) . \tag{3.32}$$

For the FC submodule, the user specifies:

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

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

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

$$\underline{\underline{A}}^{\text{fc}}_{l} \in \mathbb{R}^{N^{\text{fc}}_{\text{in},l} \times \breve{N}^{\text{fc}}_{\text{width}}}, \quad \underline{b}^{\text{fc}}_{l} \in \mathbb{R}^{\breve{N}^{\text{fc}}_{\text{width}}}. \tag{3.33}$$

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

$$N^{\text{fc}}_{\text{params}} = \left(N^{\text{fc}}_{\text{in},1} + 1 + \left(\breve{N}^{\text{fc}}_{\text{layer}} - 1\right)\left(\breve{N}^{\text{fc}}_{\text{width}} + 1\right)\right) \breve{N}^{\text{fc}}_{\text{width}}. \tag{3.34}$$

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

$$\mathcal{F}^{\text{fc}}\left(\underline{h}\right)_i, \quad i \in \left\{1, \ldots, \breve{N}^{\text{batch}}\right\}. \tag{3.35}$$

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

$$\left(\tilde{v}^{\text{d}}_{\text{des}}, {}_{\mathbf{I}}\underline{p}^{\text{wp}}\right) \tag{3.36}$$

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

$$\left({}_{\mathbf{L}}\underline{\omega}^{\text{d}}_{\text{des}}, {}_{\mathbf{L}}\underline{\dot{\omega}}^{\text{d}}_{\text{des}}, c^{\text{d}}_{\text{des}}\right) \tag{3.37}$$

comprises the desired angular velocity ${}_{\mathbf{L}}\underline{\omega}^{\text{d}}_{\text{des}}$ and acceleration ${}_{\mathbf{L}}\underline{\dot{\omega}}^{\text{d}}_{\text{des}}$ of the drone in the local reference system (see fig. 3.1a) as well as the desired collective thrust $c^{\text{d}}_{\text{des}}$ of the 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. 3.31) as it applies an activation and a biased linear transformation on the input feature vector

$$\mathcal{F}^{\text{head}} : \mathbb{R}^{N^{\text{head}}_{\text{in}}} \to \mathbb{R}^{N^{\text{head}}_{\text{out}}}; \quad \underline{x} \mapsto \underline{\underline{A}}^{\text{head}} \cdot \overset{\odot}{\breve{f}}{}^{\text{head}}\left(\underline{x}\right) + \underline{b}^{\text{head}}. \tag{3.38}$$

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

$$N_{\text{in}}^{\text{head}} = \dim\left(\underline{x}\right), \tag{3.39}$$

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

$$N_{\text{out}}^{\text{head}} = \begin{cases} 3, & \text{if navigation decision} \\ 7, & \text{if control command.} \end{cases} \tag{3.40}$$

The total number of trainable parameters of the HEAD submodule is

$$N_{\text{params}}^{\text{head}} = \left(N_{\text{in}}^{\text{head}} + 1\right) N_{\text{out}}^{\text{head}}. \tag{3.41}$$

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. 3.36) 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

$$(\tilde{v}_{\text{des}}^{\text{d}},\ {}_{\mathbf{I}}\underline{p}^{\text{wp}}). \tag{3.42}$$

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

$${}_{\mathbf{G}}\underline{p}^{\text{d}},\ {}_{\mathbf{G}}\underline{v}^{\text{d}},\ {}_{\mathbf{G}}\underline{a}^{\text{d}}. \tag{3.43}$$

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

$$v^{\text{d}}_{\text{des}} = \max\left(\breve{v}^{\text{d}}_{\text{min}},\ \breve{v}^{\text{d}}_{\text{max}} \cdot \tilde{v}^{\text{d}}_{\text{des}}\right). \tag{3.44}$$

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

2. Compute the drone's distance to the waypoint

$$d^{\text{d-wp}} = \max\left(\breve{d}^{\text{d-wp}}_{\text{min}},\ \min\left(v^{\text{d}}_{\text{des}} \cdot \breve{t}^{\text{d-wp}}_{\Delta},\ \breve{d}^{\text{d-wp}}_{\text{max}}\right)\right). \tag{3.45}$$

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

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

$$_{\mathbf{G}}\underline{p}^{\text{wp}} = T_{\mathbf{GI}}\left(d^{\text{d-wp}},\ _{\mathbf{I}}\underline{p}^{\text{wp}}\right). \tag{3.46}$$

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

$$t^{\text{lt}}_0 = t \tag{3.47}$$

and compute the duration of the local trajectory

$$t^{\text{lt}}_{\Delta} = \frac{d^{\text{d-wp}}}{\min\left(v^{\text{d}}_{\text{des}},\ \|_{\mathbf{G}}\underline{v}^{\text{d}}\|_2 + \breve{v}^{\text{d}}_{\Delta}\right)}. \tag{3.48}$$

The drone's distance $d^{\text{d-wp}}$ to the waypoint is devided by the slower of either the desired speed $v^{\text{d}}_{\text{des}}$ or the latest drone speed estimate $\left\|_{\mathbf{G}}\underline{v}^{\text{d}}\right\|_2$ plus a user-specified speed increment $\breve{v}^{\text{d}}_{\Delta}$. 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

$$_{\mathbf{G}}\underline{p}^{\text{lt}} :\ \left[0, t^{\text{lt}}_{\Delta}\right] \to \mathbb{R}^3; \quad t \mapsto {_{\mathbf{G}}\underline{p}^{\text{lt}}}(t) \tag{3.49}$$

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. [23] is deployed to find the polynomial trajectory with minimum jerk (third time derivative of position) by solving the optimization problem

$$\begin{aligned} &\min \int_0^{t_\Delta^{\mathrm{lt}}} \left\| {}_{\mathbf{G}}\dddot{\underline{p}}^{\mathrm{lt}}(t) \right\|_2^2 \mathrm{d}t \\ \text{s.t.}\quad & {}_{\mathbf{G}}\underline{p}^{\mathrm{lt}}(0) = {}_{\mathbf{G}}\underline{p}^{\mathrm{d}} \qquad && {}_{\mathbf{G}}\underline{p}^{\mathrm{lt}}(t_\Delta^{\mathrm{lt}}) = {}_{\mathbf{G}}\underline{p}^{\mathrm{wp}} \\ & {}_{\mathbf{G}}\dot{\underline{p}}^{\mathrm{lt}}(0) = {}_{\mathbf{G}}\underline{v}^{\mathrm{d}} && {}_{\mathbf{G}}\dot{\underline{p}}^{\mathrm{lt}}(t_\Delta^{\mathrm{lt}}) \text{ free} \\ & {}_{\mathbf{G}}\ddot{\underline{p}}^{\mathrm{lt}}(0) = {}_{\mathbf{G}}\underline{a}^{\mathrm{d}} && {}_{\mathbf{G}}\ddot{\underline{p}}^{\mathrm{lt}}(t_\Delta^{\mathrm{lt}}) \text{ free.} \end{aligned} \tag{3.50}$$

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

$$t^{\mathrm{lt}} = t - t_0^{\mathrm{lt}} + 1/\breve{f}^{\mathrm{main}}. \tag{3.51}$$

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

$$\begin{aligned} {}_{\mathbf{G}}\underline{p}^{\mathrm{ref}} &= {}_{\mathbf{G}}\underline{p}^{\mathrm{lt}}(t^{\mathrm{lt}}) \\ {}_{\mathbf{G}}\underline{v}^{\mathrm{ref}} &= {}_{\mathbf{G}}\dot{\underline{p}}^{\mathrm{lt}}(t^{\mathrm{lt}}) \\ {}_{\mathbf{G}}\underline{a}^{\mathrm{ref}} &= {}_{\mathbf{G}}\ddot{\underline{p}}^{\mathrm{lt}}(t^{\mathrm{lt}}) \\ {}_{\mathbf{G}}\underline{j}^{\mathrm{ref}} &= {}_{\mathbf{G}}\dddot{\underline{p}}^{\mathrm{lt}}(t^{\mathrm{lt}}) \\ \phi_z^{\mathrm{ref}} &= \operatorname{atan2}\left({}_{\mathbf{G}}v_y^{\mathrm{ref}}, {}_{\mathbf{G}}v_x^{\mathrm{ref}}\right). \end{aligned} \tag{3.52}$$

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

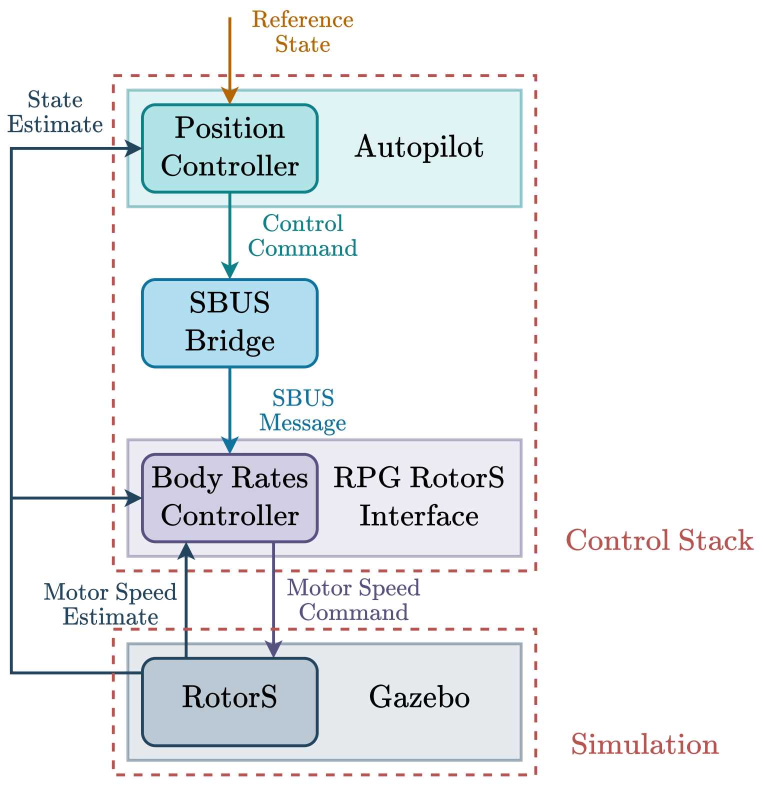


Figure 3.3: The control stack in simulation.

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. 3.52) 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 [6]. 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 rotors as

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

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 [5]. 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. [8], 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 [16].

Problem
This thesis implements the expert system by Kaufmann et al. [18] 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

$$ {}_{\mathrm{G}}\underline{p}_{\mathrm{wp}}^{\mathrm{ann}} \tag{3.53}$$

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

with respect to the global reference system (see equ. 3.46) that was computed based on the ANN module's latest navigation decision (see equ. 3.36).

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

$$_{\mathbf{G}}\underline{p}^{\mathrm{d}}, \quad _{\mathbf{G}}\underline{q}^{\mathrm{d}}. \tag{3.54}$$

F3 The center points of the gates of the racetrack

$$\left(_{\mathbf{G}}\underline{p}_i^{\mathrm{gate}}\right)_{i\in\{0,\dots,N^{\mathrm{gate}}-1\}} \tag{3.55}$$

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

$$i_{\mathrm{target}}^{\mathrm{gate}} \in \{0, \dots, N^{\mathrm{gate}} - 1\}. \tag{3.56}$$

R1 Compute the global trajectory of the current racetrack

$$_{\mathbf{G}}\underline{p}^{\mathrm{gt}} : \left[t_0^{\mathrm{gate}}, t_{N^{\mathrm{gate}}}^{\mathrm{gate}}\right] \to \mathbb{R}^3; \quad t \mapsto {_{\mathbf{G}}\underline{p}^{\mathrm{gt}}}(t). \tag{3.57}$$

The algorithm of Mellinger and Kumar [21] finds the minimum snap (fourth time derivative of position) spline trajectory

$$_{\mathbf{G}}\underline{p}^{\mathrm{gt}}(t) = \sum_{i=0}^{N^{\mathrm{gate}}-1} \begin{cases} _{\mathbf{G}}\underline{p}_i^{\mathrm{gt}}(t), & t \in \left[t_i^{\mathrm{gate}}, t_{i+1}^{\mathrm{gate}}\right] \\ 0, & \text{else} \end{cases} \tag{3.58}$$

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

$$\begin{aligned} &\underset{_{\mathbf{G}}\underline{p}^{\mathrm{gt}}}{\operatorname{argmin}} \int_{t_0^{\mathrm{gate}}}^{t_{N^{\mathrm{gate}}}^{\mathrm{gate}}} \left\| _{\mathbf{G}}\ddddot{\underline{p}}^{\mathrm{gt}}(t) \right\|_2^2 \mathrm{d}t \\ &\text{s.t.} \quad _{\mathbf{G}}\underline{p}^{\mathrm{gt}}\left(t_i^{\mathrm{gate}}\right) = {_{\mathbf{G}}\underline{p}_i^{\mathrm{gate}}}, \qquad \frac{\mathrm{d}^j {_{\mathbf{G}}\underline{p}^{\mathrm{gt}}}}{\mathrm{d}t^j}\left(t_i^{\mathrm{gate}}\right) \text{ defined}, \\ &\quad\quad i \in \{0, \dots, N^{\mathrm{gate}}\}, \qquad j \in \left\{1, \dots, \breve{N}_{\mathrm{cont}}^{\mathrm{gt}}\right\}. \end{aligned} \tag{3.59}$$

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

$$\begin{aligned} &\underset{t^{\text{gate}}_i}{\operatorname{argmin}} \int_{t^{\text{gate}}_0}^{t^{\text{gate}}_{N^{\text{gate}}}} \left\| {}_{\mathbf{G}}\dddot{\underline{p}}^{\text{gt}}(t) \right\|_2^2 \mathrm{d}t \\ \text{s.t.}\quad & t^{\text{gate}}_i < t^{\text{gate}}_{i+1}, \qquad t^{\text{gate}}_0,\ t^{\text{gate}}_{N^{\text{gate}}} \text{ fixed}, \qquad i \in \{0, ..., N^{\text{gate}} - 1\} \end{aligned} \tag{3.60}$$

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

$$\left({}_{\mathbf{G}}\underline{p}^{\text{gt}}_i\right)_{i\in\{0,...,N^{\text{gt}}-1\}}, \quad \left({}_{\mathbf{G}}v^{\text{gt}}_i\right)_{i\in\{0,...,N^{\text{gt}}-1\}} \tag{3.61}$$

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

$$\left\| {}_{\mathbf{G}}\underline{p}^{\text{gate}}_{i^{\text{gate}}_{\text{target}}} - {}_{\mathbf{G}}\underline{p}^{\text{d}} \right\|_2 < \breve{d}^{\text{d-gate}}, \tag{3.62}$$

increment the index to the currently targeted gate

$$i^{\text{gate}} \leftarrow \left(i^{\text{gate}} + 1\right) \bmod N^{\text{gate}}. \tag{3.63}$$

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.,

$$\underset{i\in\{0,...,N^{\text{gt}}-1\}}{\text{argmin}} \left\| {}_{\mathbf{G}}\underline{p}^{\text{ann}}_{\text{wp}} - {}_{\mathbf{G}}\underline{p}^{\text{gt}}_{i} \right\|_2 > \breve{d}^{\text{wp-gt}}_{\text{max}} \cdot \frac{\breve{v}^{\text{d}}_{\text{max}} + 1}{5}, \tag{3.64}$$

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

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

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

$$i^{\text{gt}}_{\text{prev}} = (i^{\text{gt}}_{\text{proj}} - 1 + N^{\text{gt}}) \bmod N^{\text{gt}}. \tag{3.65}$$

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

$${}_{\mathbf{G}}\underline{a} = {}_{\mathbf{G}}\underline{p}^{\text{gt}}_{i^{\text{gt}}_{\text{proj}}} - {}_{\mathbf{G}}\underline{p}^{\text{gt}}_{i^{\text{gt}}_{\text{prev}}} \tag{3.66}$$

and the vector to the current drone position

$${}_{\mathbf{G}}\underline{b} = {}_{\mathbf{G}}\underline{p}^{\text{d}} - {}_{\mathbf{G}}\underline{p}^{\text{gt}}_{i^{\text{gt}}_{\text{prev}}}. \tag{3.67}$$

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

$$\frac{{}_{\mathbf{G}}\underline{a} \cdot {}_{\mathbf{G}}\underline{b}}{{}_{\mathbf{G}}\underline{a} \cdot {}_{\mathbf{G}}\underline{a}} < 1, \tag{3.68}$$

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

$$i^{\text{gt}}_{\text{proj}} \leftarrow (i^{\text{gt}}_{\text{proj}} + 1) \bmod N^{\text{gt}} \tag{3.69}$$

and go back to step 1.

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

$$\left\| {}_{\mathbf{G}}\underline{p}^{\text{d}} - {}_{\mathbf{G}}\underline{p}^{\text{gt}}_{i^{\text{gt}}_{\text{proj}}} \right\|_2 \leq \breve{d}^{\text{d-proj}}, \tag{3.70}$$

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

$$\underset{i^{\text{gt}}_{\text{proj}}}{\text{argmin}} \left\| {}_{\mathbf{G}}\underline{p}^{\text{d}} - {}_{\mathbf{G}}\underline{p}^{\text{gt}}_{i^{\text{gt}}_{\text{proj}}} \right\|_2. \tag{3.71}$$

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

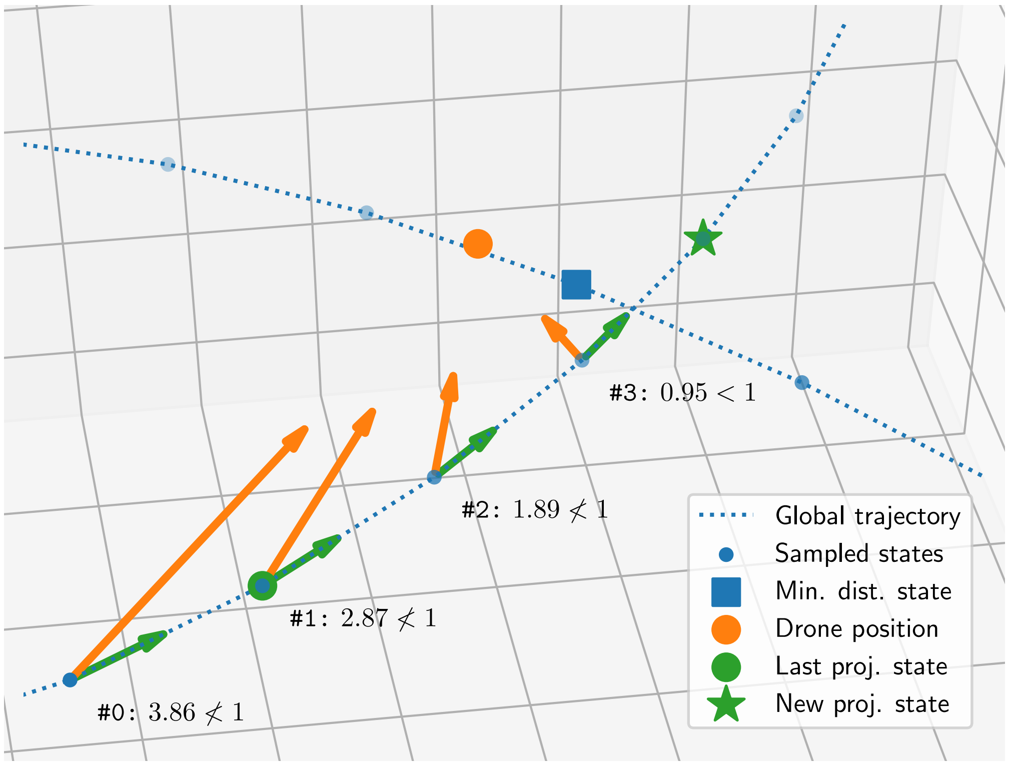


Figure 3.4: Schematic example of the update of the projection state index (**R4**). Known are: the positions (blue points) sampled from the global trajectory (blue dotted line), the last index to the projection state (green circle) and the current position of the drone (orange circle). At an iteration, the vector from the previous to the projection state (green arrows, equ. 3.66) and the vector from the previous to the drone position (orange arrows, equ. 3.67) are computed. Then, the normalized dot product criterion (annotations, equ. 3.68) 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. 3.70) is also met). Note that finding the index to the projection state only with minimum distance (equ. 3.71) 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
$$i_v^{\text{gt}} = i_{\text{proj}}^{\text{gt}}. \tag{3.72}$$
2. Increment the searched index
$$i_v^{\text{gt}} \leftarrow (i_v^{\text{gt}} + 1) \bmod N^{\text{gt}}. \tag{3.73}$$
3. If the speed state is further from the projection state than a user-specified distance
$$\left\| {}_{\mathbf{G}}\underline{p}_{i_v^{\text{gt}}}^{\text{gt}} - {}_{\mathbf{G}}\underline{p}_{i_{\text{proj}}^{\text{gt}}}^{\text{gt}} \right\|_2 > \breve{d}^{\text{proj-}v}. \tag{3.74}$$
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
$$d^{\text{proj-wp}} = \max\left(\breve{d}_{\text{min}}^{\text{proj-wp}}, \operatorname*{argmin}_i \left\| {}_{\mathbf{G}}\underline{p}_i^{\text{gate}} - {}_{\mathbf{G}}\underline{p}^{\text{d}} \right\|_2 \right), \tag{3.75}$$
$$i \in \left\{ i_{\text{target}}^{\text{gate}}, (i_{\text{target}}^{\text{gate}} - 1 + N^{\text{gate}}) \bmod N^{\text{gate}} \right\}. \tag{3.76}$$
2. Initialize the searched index with the index to the projection state
$$i_{\text{wp}}^{\text{gt}} = i_{\text{proj}}^{\text{gt}}. \tag{3.77}$$
3. Increment the searched index
$$i_{\text{wp}}^{\text{gt}} \leftarrow (i_{\text{wp}}^{\text{gt}} + 1) \bmod N^{\text{gt}}. \tag{3.78}$$
4. If the waypoint state is further from the projection state than the distance computed in step 1
$$\left\| {}_{\mathbf{G}}\underline{p}_{i_{\text{wp}}^{\text{gt}}}^{\text{gt}} - {}_{\mathbf{G}}\underline{p}_{i_{\text{proj}}^{\text{gt}}}^{\text{gt}} \right\|_2 > d^{\text{proj-wp}}, \tag{3.79}$$
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

$$\tilde{v}^{\text{exp}}_{\text{des}} = \frac{{}_{\mathbf{G}}v^{\text{gt}}_{i^{\text{gt}}_{v}}}{\underset{i\in\{0,\ldots,N^{\text{gt}}-1\}}{\text{argmax}} \left\| {}_{\mathbf{G}}v^{\text{gt}}_{i} \right\|_2} \in [0,1]. \tag{3.80}$$

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

$$_{\mathbf{I}}\underline{p}^{\text{exp}}_{\text{wp}} = T_{\mathbf{IG}} \left({}_{\mathbf{I}}\underline{p}^{\text{gt}}_{i^{\text{gt}}_{\text{wp}}} \right). \tag{3.81}$$

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

4.1 Simulation Setup

The experiments of this thesis are conducted in a drone racing simulation, which includes the drone, the gates constituting the racetrack and the scenery. The implementation of the simulation is devided into physics modelling and image rendering (see figure 4.1).

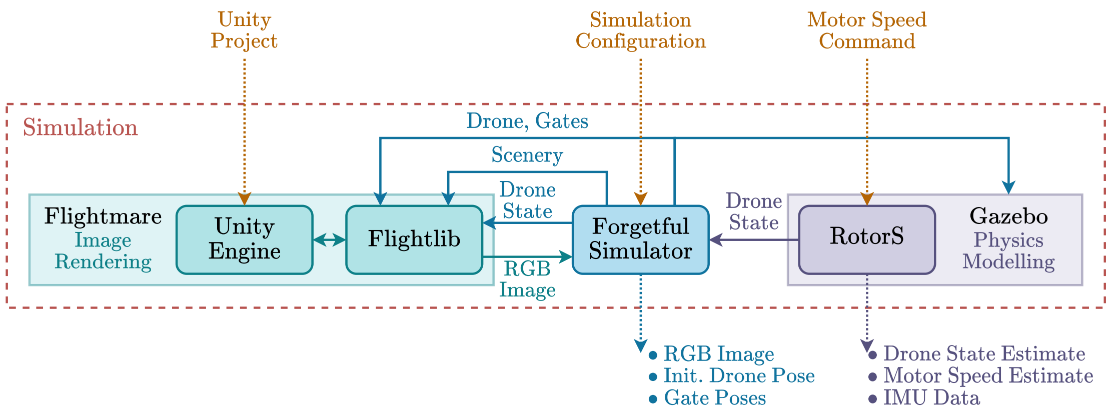


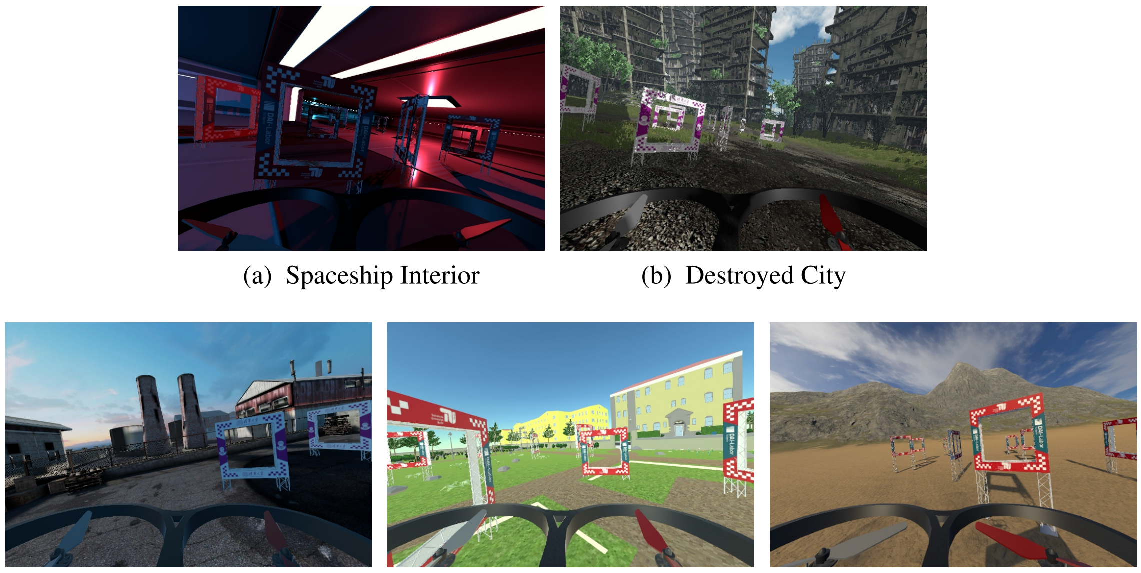
Figure 4.1: Implementation concept of the simulation

The Gazebo[1] simulator is deployed to model the physics with high accuracy. This includes the drone's dynamics and potential collisions of the drone with the gates. The RotorS [8] plugin actuates the drone model with the inputted motor speed commands

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

and outputs the drone state estimate, the motor speed estimate and the data from the drone's IMU unit.

The Flightmare [29] simulator is deployed to render images that are almost photo-realistic. At a user-specified frequency, the Flightlib interface updates the drone's pose, and therewith its onboard camera, within the Unity[2] Engine and fetches an RGB image from the onboard camera. The Unity Engine is built from a Unity project that integrates the functionalities of the RPG Flightmare Unity Project[3]. The Unity project of this thesis and entails five scenes named spaceship interior[4], destroyed city[5], industrial site[6], polygon city[7] and desert mountain[8] (see figure 4.2). Each scene has three sites (A, B, C) to locate a racetrack. Two different gate covers, one with TU Berlin/DAI-Labor and one with Tsinghua University/DME logos, are available (see figure 4.3).



(a) Spaceship Interior (b) Destroyed City

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

Figure 4.2: Scenes implemented in simulation

The Forgetful Simulator ROS node takes on three tasks. First, it synchronizes the drone state in the Flightmare simulator with the ground-truth state in the Gazebo simulator. Second, it fetches the RGB images from the Flightmare simulator and makes

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

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

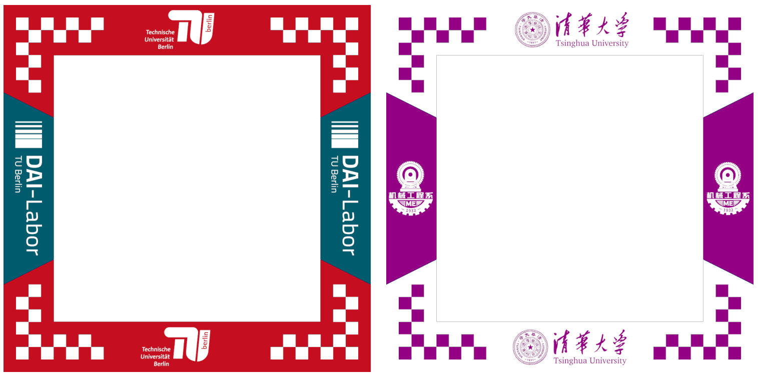
[4]based on the ”3D Free Modular Kit” from the Unity Asset Store

[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: Gate covers implemented in simulation

them available as output. Third, it setups the simulation as requested by the inputted simulation configuration.

The simulation configuration specifies the scene (spaceship interior, destroyed city, industrial site, polygon city or desert mountain) , the site (A, B, C), the gate cover (TUB-DAI or THU-DME) and the racetrack configuration. The racetrack configuration, in turn, specifies the type (figure-8 or gap), the generation (deterministic or randomized) and the direction (clockwise or counterclockwise). The Forgetful Simulator, which stores the data of the deterministic, counterclockwise gate poses of the available racetrack types (see table 4.1), processes the simulation configuration as shown in figure 4.4.

If it was specified, randomize the stored gate poses of the specified racetrack type with the following steps.

1. This step applies only to the gap racetrack type as it explicitely randomizes the gap distance. Sample the y-position values of the gates #3 -6 from the uniform real distribution over the intervals specified in table 4.1.

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 all gate positions by the same value sampled 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 $\left[-\breve{d}^{\text{sim}}_{\text{twist,max}}, \breve{d}^{\text{sim}}_{\text{twist,max}}\right]$.

Then, if it was specified, redirect the gate poses from counterclockwise to clockwise. Compute the drone's starting pose to face towards the last gate and to be located between the second last and the last gate. The Forgetful Simulator loads the specified scene and site in the Flightmare Simulator and spawns the drone model and the gate models (with the specified cover) at the computed poses in both, the Flightmare and the Gazebo, simulator. Finally, the Forgetful Simulator outputs the computed initial drone and gate poses, which are required by, e.g., the expert system.

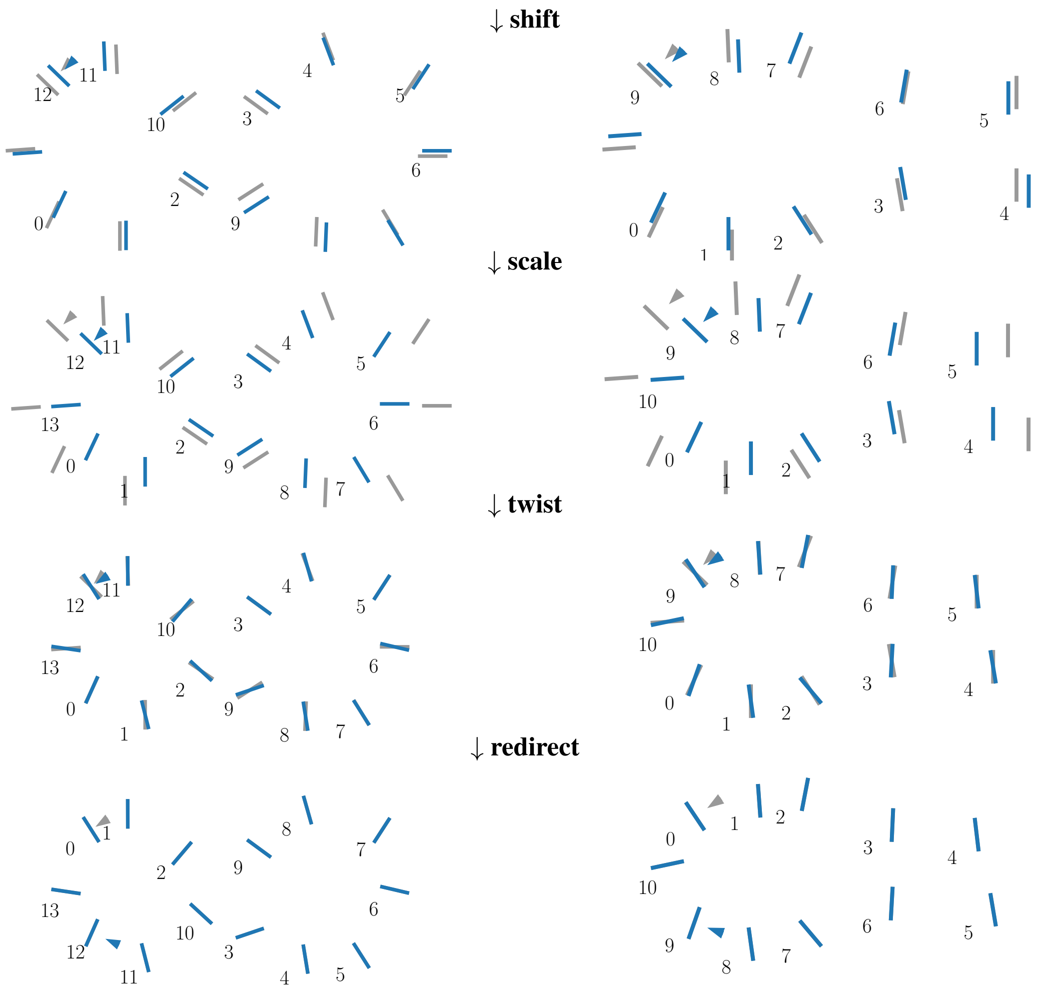


Figure 4.4: Computation of the racetrack for the figure-8 (left) and the gap (right) racetrack type. In the xy-plane, the drone's starting pose (arrow) and the gate poses (numbered bars) are depicted before (grey) and after (blue) the individual processing step. The randomizing processing step exclusive to the gap racetrack is not shown.

Table 4.1: Deterministic gate poses

racetrack	gate	x	y	z	yaw
	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
Figure-8	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
	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
Gap	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 ANN Module Variants

In the experiments, several variants of the ANN module (see section 3.2) are examined for their performance. Table 4.2 shows the variant configurations with respect to the input, output and the CNN, GRU, FC, and HEAD submodule. The input configuration includes: the sequence length $\breve{N}^{\text{seq}}$ of the samples the variant is trained on, the resize factor $\breve{s}^{\text{cnn}}_{\text{resize}}$ of the image preprocessing (see equ. ??) as well as the switchable optional inputs t^{cam}_{Δ}, $({}_{\mathbf{L}}\underline{\hat{a}}^{\text{imu}}, {}_{\mathbf{L}}\underline{\hat{\omega}}^{\text{imu}})$ and t^{imu}_{Δ} (see ...). The output configuration includes the two options of navigation decisions $(\tilde{v}^{\text{d}}_{\text{des}}, {}_{\mathbf{I}}\underline{p}^{\text{wp}})$ and control commands $({}_{\mathbf{L}}\underline{\omega}^{\text{d}}_{\text{des}}, {}_{\mathbf{L}}\underline{\dot{\omega}}^{\text{d}}_{\text{des}}, c^{\text{d}}_{\text{des}})$ (see section ...). For the CNN, GRU, FC and HEAD configuration see the corresponding paragraphs in section 3.2.

All of the examined variants have three configurational aspects in common. First, all available optional inputs are activated. Second, navigation decisions are selected as output option. Third, the CNN submodule implements the backbone of the ResNet18 PyTorch implementation whose parameters are trainable and pretrained on ImageNet !!!. The ResNet18 contains 18 layers and has ... trainable parameters ...

Table 4.2: ANN module variants

		Baseline	Baseline+	Sequential
Input	$\breve{N}^{\text{seq}}$	1	1	$\{2, 3, 5, 10, 25\}$
	$\breve{s}^{\text{cnn}}_{\text{resize}}$	½	½	$\{1/2, 1/3\}$
	t^{cam}_{Δ}	✗	✓	✓
	$\left({}_{\mathbf{L}}\underline{\hat{a}}^{\text{imu}}, {}_{\mathbf{L}}\underline{\hat{\omega}}^{\text{imu}}\right)$	✗	✓	✓
	t^{imu}_{Δ}	✗	✓	✓
Output	$\left(\tilde{v}^{\text{d}}_{\text{des}}, {}_{\mathbf{I}}\underline{p}^{\text{wp}}\right)$	✓	✓	✓
	$\left({}_{\mathbf{L}}\underline{\omega}^{\text{d}}_{\text{des}}, {}_{\mathbf{L}}\underline{\dot{\omega}}^{\text{d}}_{\text{des}}, c^{\text{d}}_{\text{des}}\right)$	✗	✗	✗
CNN	Model	resnet8	resnet18	resnet18
	Pretrained	✗	✓	✓
	Trainable	✓	✗	✓
GRU	$\breve{N}^{\text{gru}}_{\text{layer}}$	✗	✗	3
	$\breve{N}^{\text{gru}}_{\text{hidden}}$	✗	✗	16
	$\breve{p}^{\text{gru}}$	✗	✗	0.5
FC	$\breve{N}^{\text{fc}}_{\text{layer}}$	1	3	✗
	$\breve{N}^{\text{fc}}_{\text{width}}$	256	32	✗
	$\breve{p}^{\text{fc}}$	0.5	0.2063	✗
	$\breve{f}^{\text{fc}}$	ReLU	ReLU	ReLU
HEAD	$\breve{f}^{\text{head}}$	ReLU	ReLU	ReLU

4.3 Data Generation and Training

The

4.4 Racing Tests

Table 4.3: ANN module variants: number of trainable and non-trainable parameters and number of multiply-accumulate operations

	#	Baseline	Baseline+	Sequential
CNN	Trainables		0	0
	Non-trainables		11,176,512	11,689,512
	MAC Operations		1,408,910,720	
CAT	Trainables	0	0	0
	Non-trainables	0	0	0
	MAC Operations	0	0	0
GRU	Trainables	0	0	
	Non-trainables	0	0	
	MAC Operations	0	0	
FC	Trainables		41,664	0
	Non-trainables		0	0
	MAC Operations		41,664	0
HEAD	Trainables		195	
	Non-trainables		0	
	MAC Operations		195	
Total	Trainables		41,859	
	Non-trainables		11,176,512	
	MAC Operations		1,408,952,579	

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.

5.1 Results

5.2 Discussions

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] 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.

[2] 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).

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

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

[15] 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).

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

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

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

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

[20] 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.

[21] 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.

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

[23] 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.

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

[25] 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.

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

[27] 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.

[28] 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).

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

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

[31] 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 Verschlüsselungsverfahren)
ASCII	American Standard Code for Information Interchange (Computer-Textstandard)
dpi	dots per intch (Punkte pro Zoll; Maß für Auflösung 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- einschließlich Audiokompression)
MP3	MPEG-1 Audio Layer 3 (Audiokompressionformat)
PACS	Picture Archiving and Communication Systems
PNG	Portable Network Graphics (Grafikformat)
RSA	Rivest, Shamir, Adleman (asymmetrisches Verschlüsselungsverfahren)
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.

```
class Beispiel{

  public static void main(String args[]){

    System.out.println("Hello␣World");

  }

}
```

```
class Beispiel{

  public static void main(String args[]){

    System.out.println("Hello World");

  }

}
```

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