

# Bachelorarbeit

Jan Robert Rösler

## Bildbasierte Navigation mit Neuronalen Netzen

Jan Robert Rösler

# Bildbasierte Navigation mit Neuronalen Netzen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung  
im Studiengang Bachelor of Science Technische Informatik  
am Department Informatik  
der Fakultät Technik und Informatik  
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel  
Zweitgutachter: Prof. Dr. rer.nat. Stephan Pareigis

Eingereicht am: 2019

**Jan Robert Rösler**

**Thema der Arbeit**

Bildbasierte Navigation mit Neuronalen Netzen

**Stichworte**

CNN...

**Kurzzusammenfassung**

In dieser Bachelorarbeit soll untersucht werden, wie Navigation auf reinen Bilddaten funktioniert. Konkret geht es um das Erkennen einer Fahrbahn mit einem Neuronalen Netz, bzw. um das Erzeugen von Lenkwinkeldaten auf Basis eines Bildes einer Fahrbahn. Hierzu wird ein trainiertes Neuronales Netz mittels Fine Tuning abgestimmt. Das wird direkt zur Anwendung gebracht auf einem RC Fahrzeug aus dem "Carolo-Cup", inklusive Fahrten auf einer Teststrecke.

**Jan Robert Rösler**

**Title of Thesis**

Image based navigation with Neural Networks

**Keywords**

CNN...

**Abstract**

This thesis is concerned with the task of navigation, based on image data. The goal is to **todo**

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>v</b>
<b>Tabellenverzeichnis</b>	<b>vi</b>
<b>Abkürzungen</b>	<b>viii</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Autonome Navigation mit Bilddaten . . . . .	2
<b>2 Neuronale Navigation mit Bilddaten</b>	<b>3</b>
2.1 Relevante Technik/Hintergrund . . . . .	3
2.1.1 Deep Learning . . . . .	3
2.1.2 CNN . . . . .	3
2.1.3 ResNet . . . . .	4
2.1.4 Fine tuning . . . . .	5
2.2 Ansätze . . . . .	6
2.2.1 ALVINN . . . . .	6
2.2.2 NVIDIA DAVE-2 . . . . .	6
<b>3 Idee</b>	<b>9</b>
3.1 DroNet und Carolo-Cup . . . . .	9
<b>4 Entwurf</b>	<b>12</b>
4.1 Hardware und Strecke . . . . .	13
4.2 Trainingsdaten . . . . .	14
4.3 Vorverarbeitung und Fine Tuning . . . . .	15
4.3.1 Software . . . . .	15
4.3.2 Preprocessing . . . . .	16
4.3.3 Fine Tuning . . . . .	19
4.4 Steuerung . . . . .	22

<b>5</b>	<b>Auswertung und Szenarien</b>	<b>24</b>
5.1	Training . . . . .	24
5.2	Testfahrt . . . . .	25
5.3	Einzel szenarien . . . . .	27
5.4	Saliency . . . . .	29
<b>6</b>	<b>Resümee</b>	<b>31</b>
	<b>Glossar</b>	<b>36</b>
	<b>Selbstständigkeitserklärung</b>	<b>38</b>

# Abbildungsverzeichnis

2.1	Residual Block . . . . .	4
2.2	ALVINN Architektur (a) und simulierte Fahrbahn (b) . . . . .	7
2.3	Komponenten des Trainings . . . . .	8
3.1	Architektur DRONET . . . . .	9
3.2	Einfacher schematischer Aufbau . . . . .	11
4.1	Das Carolo-Cup Fahrzeug . . . . .	13
4.2	Ein Ausschnitt der Teststrecke . . . . .	14
4.3	Kameraaufnahme der Fahrbahn . . . . .	15
4.4	Skalierung der Lenkwinkel-Wertebereiche . . . . .	16
4.5	Lenkwinkelverteilung im Bilderset . . . . .	17
4.6	Bildverarbeitungsschritte . . . . .	18
4.7	Beispiele verschiedener Bearbeitungsstufen, zugeschnitten (a), nach Histogrammausgleich (b) und nach Rausch-Filter (c) . . . . .	19
4.8	Angepasste Architektur . . . . .	20
4.9	Struktur des Lernprozesses . . . . .	21
4.10	Komponentenstruktur und Verbindung der Neuronalen Steuerung . . . . .	23
5.1	Traningsfehler (train) und Validierungsfehler (val) über 100 Epochen . . . . .	24
5.2	Fehlerszenarien . . . . .	27
5.3	Kreuzungsszenario aus Sicht der Fahrzeugkamera . . . . .	28
5.4	Ausschlaggebende Bildbereiche . . . . .	30

# Tabellenverzeichnis

5.1	Algorithmen im Vergleich . . . . .	26
5.2	Vergleich der Autonomie . . . . .	26

# Abkürzungen

**ALVINN** Autonomous Land Vehicle In a Neural Network.

**IDS** Imaging Development Systems GmbH.

**NUC** Next Unit of Computing.

**UDS** Unix Domain Sockets.

**VSCode** Virtual Studio Code.



# 1 Einleitung

Sobald ein System, welcher Art sei offen, mobil wird, also läuft, rollt, gleitet, schwebt oder schwimmt, steht es vor der Aufgabe der Navigation. Das kann zunächst bedeuten, zu Wissen, wo es sich befindet. Auf einer Karte oder auch relativ zu anderen Objekten in der Umgebung. In der Robotik nennt man diese Kompetenz Lokalisation. Wenn man weiß wo man ist, könnte sich zusätzlich die Frage stellen, wie man zu einem bestimmten Ort hinkommt. Je nach Ziel oder Aufgabe des mobilen Systems, ist die in der Robotik Pfadplanung genannte Kompetenz von Bedeutung.

Um sich in einer Umgebung zu bewegen, kann es ebenfalls interessant sein, eine eigene Repräsentation der Umgebung zu erstellen. Das Aufbauen einer Karte ist in der Robotik das Mapping und beinhaltet als Disziplin auch das Interpretieren und Auswerten von den gesammelten Umgebungsinformationen in einer Karte.

Es wird vorausgesetzt, dass ein solches System bzw. ein solcher Roboter keinerlei Navigationshilfe (z.B. Steuersignale) von außen erfährt, sondern völlig autonom Entscheiden und navigieren muss.

Als Grundstein der Entwicklung solcher autonomer Systeme kann man die Erfindung von dem Neurophysiologen William Grey Walter festmachen [10]. Seine in den späten 40er Jahren entwickelten und „Elmer“ und „Elsie“ getauften Roboter, sollten ihm dabei helfen, das menschliche Gehirn besser zu verstehen. Beide Roboter, wegen ihrer Form auch Schildkröten genannt, konnten mit je einem Licht- und Berührungssensor, die wiederum jeweils mit einem Motor verbunden waren, unter anderem um Hindernisse herum navigieren. Die getrennte Ansteuerung der Motoren sollte Neuronen im Gehirn simulieren. Diese Gehirnanalogie findet sich in moderner Weise in dieser Arbeit wieder, Neuronale Netze werden im Folgenden eine zentrale Rolle spielen.

Navigation in der Robotik setzt sich also aus verschiedenen Unterdisziplinen zusammen, die in ihrem Zusammenspiel ganz besonders aktuell viel Beachtung finden: Fahrzeuge der Firma Tesla sind bereits auf öffentlichen Straßen autonom unterwegs. Teil der technischen Ausstattung der Fahrzeuge (neben Ultraschallsensoren und Radar) sind Kameras zur Erfassung der Umgebung.

In dieser Arbeit wird gerade die visuelle, bildbasierte Navigation die zweite zentrale Rolle haben.

### 1.1 Autonome Navigation mit Bilddaten

Legt man der Navigationsaufgabe als einzigen Lösungsraum die Bilddaten einer Kamera zugrunde, dann bieten sich verschiedene Möglichkeiten an, diese zu verwenden. Objekterkennung ist häufig der erste Schritt, um sich in einer Umgebung zu orientieren und herauszufinden, aus was sich diese Umgebung überhaupt zusammensetzt. Zusätzlich muss entschieden werden, welche Teile dieser aus Objekten und Flächen aufgebauten Umwelt überhaupt befahrbar ist. Für Systeme, die in Szenarien verwendet werden sollen, in denen auf Fahrbahnen, Straßen oder andersartig begrenzten Strecken navigiert werden soll, ist die Erkennung dieser Strecke die Schlüsselfunktion. In diesen Fällen bestimmt die Umgebung direkt, wo überhaupt gefahren werden darf, für die Navigation eine enorme Hilfe.

Um genau solche Szenarien soll es in dieser Arbeit gehen, speziell um das Fahren auf einer markierten Fahrbahn, die durch eine Kamera am Fahrzeugsystem erfasst wird.

## 2 Neuronale Navigation mit Bilddaten

### 2.1 Relevante Technik/Hintergrund

Hier erfolgt zunächst eine kurze Beschreibung der für Neuronale Navigation auf Bilddaten relevanten Technik. Grundlegendes wird nur der Vollständigkeit halber erwähnt, speziellere Aspekte kurz vorgestellt.

#### 2.1.1 Deep Learning

Im Bereich des Maschinellen Lernens stellt das Deep Learning einen besonders populären Subbereich dar. Viele bekannte Anwendungen, die in den Medien auch als Künstliche Intelligenz bezeichnet werden, nutzen diese Technik. Eine Hierarchie aus Konzepten zu bauen, bei denen komplizierte Konzepte aus dem Produkt vieler einfacher Konzepte entstehen, erwies sich für das Lernen von Datenrepräsentationen als besonders geeignet. Diese Aufeinandererschichten von Berechnungsschritten in so genannten Layern, bildet ein tiefer Graph ab, auch Neuronales Netz genannt. Die Bezeichnung Deep Learning bezieht sich auf diese Struktur, mit der das Wissen aus Erfahrungswerten, den Trainingsdaten, extrahiert werden kann.

Deep Learning gliedert sich selbst in viele Unterthemen auf, eine Darstellung der Geschichte und Technikentwicklung soll hier nicht gegeben werden. Basiswissen zu maschinellem Lernen wird vorausgesetzt, Deep Learning im Speziellen wird in entsprechender Literatur detailliert erklärt, für diese Arbeit diente vor allem ein Hauptbuch ([3]) als begleitende Wissenquelle.

#### 2.1.2 CNN

*Convolutional Neural Networks*, oder kurz CNN, haben sich gerade im Bereich der Bildverarbeitung als überlegen bewiesen. Aus der Biologie inspiriert, findet hier besonders

das Prinzip des rezeptiven Feldes Anwendung. Die Aktivität jedes Neurons wird mithilfe einer Faltung berechnet, räumliche Informationen und Zusammenhänge werden so besser erhalten. 2012 konnte ein CNN, AlexNet [6], beim ImageNet-Wettbewerb den Benchmark-Rekord von 25,8 % auf 16,4 % drücken, seitdem sind alle gut platzierten Modelle CNNs. Nicht nur in der Bilderkennung, sondern auch in der Spracherkennung ( Parsen, Satzmodellierung, Maschinelle Übersetzung) gelten CNNs als State-of-the-Art Methode und finden Anwendung in modernsten Technologien.

CNNs sollen hier nicht detailliert erklärt werden, das Verständnis wird als Grundlage vorausgesetzt.

### 2.1.3 ResNet

*Residual Neural Networks*, oder kurz ResNet, ist eine weitere Technik, die ihren Ursprung in der Biologie hat. Das Verhalten so genannter Pyramidenzellen, Nervenzellen im menschlichen Gehirn, wird nachgebildet, indem Abkürzungen zwischen Layer eingebaut werden. Residual Netze wurden 2015 entwickelt, insbesondere ließ sich mit diesem Ansatz das Trainieren tiefer Netze verbessern [4]. Abbildung 2.1 zeigt den schematischen Aufbau eines so genannten Residual Blocks.

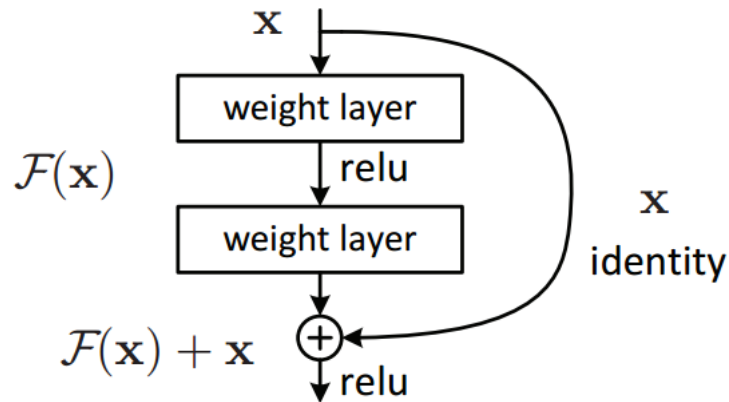


Abbildung 2.1: Residual Block  
Quelle: [13]

Je tiefer ein Neuronales Netz ist, desto schwieriger wird das Trainieren aller Layer, ein Problem in diesem Zusammenhang ist das „Vanishing Gradient Problem“. Für tiefe Layer wird der berechnete Gradient so verschwindend klein, dass irgendwann keine Änderung der Gewichte in den Layern mehr stattfindet, so stagniert das Training und es können

keine weiteren Fortschritte erzielt werden.

Der Residual-Ansatz fügt einem normalen Block gewichteter Layer eine Abkürzung hinzu. Wie in Abbildung 2.1 zu sehen, wird der Input  $x$  des Blocks direkt auf den Output addiert. Nimmt man an, dass es für den Input  $x$  eine richtige Gewichtsverteilung  $H(x)$  der Layer im Block gibt, die erlernt werden soll, dann ist der Unterschied zwischen dem Input und dieser optimalen Verteilung  $F(x)=H(x)-x$ .  $F(x)$  ist das „Residual“, stellt man die Funktion um, erhält man  $H(x)=F(x)+x$ . Statt der optimalen Gewichtsverteilung für den Block, lernt das Netz jetzt  $F(x)$ , also gerade den Unterschied zur Identitätsfunktion des Inputs  $x$ .

Die Annahme der Autoren ([4]) war, dass diese Residual-Funktion besser zu erlernen sei, als das originale Mapping. Selbst wenn das gelernte Residual  $F(x)=0$ , bekommt man für den Input  $x$  immernoch die Identitätsfunktion, also  $x$  selbst heraus. Das dieses Mapping meistens näher an der gewünschten optimalen Gewichtsverteilung liegt, war eine weitere Annahme.

In Versuchen mit Neuronalen Netzen verschiedener Tiefe konnten diese Annahmen bestätigt werden.

### 2.1.4 Fine tuning

Fine Tuning beschreibt einen Prozess, in dem ein bereits trainiertes Neuronales Netz für eine ähnliche Aufgabe abgestimmt wird. Dabei können Teile der Layer des ursprünglichen Netzes ersetzt werden, oder neue hinzugefügt werden. Insbesondere bei Klassifizierungsaufgaben wird dazu häufig die Kategorisierung verändert, um zum Beispiel die von einem Netz erlernten Eigenschaften auf neue Kategorien anzuwenden.

In diesem Zusammenhang wird auch „Layer-Freezing“, also das Einfrieren von Layern vor dem Training verwendet. Eingefrorene Layer aktualisieren ihre Gewichte nicht und sind damit vom Trainingsprozess ausgeschlossen. So soll verhindert werden, dass bereits erlernte Eigenschaften eines Netzes, wie „Katzenohr“ oder auch „Linie“ nicht verloren gehen. Dazu werden frühere Layer meist eingefroren, spätere auf einem neuen Datensatz trainiert, um die bereits erlernten Eigenschaften zu nutzen und nicht zu verändern und Eigenschaften für die neue, ähnliche Aufgabe zu erlernen.

### 2.2 Ansätze

Im folgenden wird auf zwei Ansätze der Navigation mit Neuronalen Netzen eingegangen, durch Gegenüberstellung erster Versuche mit einem modernen Ansatz soll folgenden Ausarbeitungen ein Rahmen gegeben werden.

#### 2.2.1 ALVINN

Versuche durch neuronale Verarbeitung von reinen Bilddaten in einem Szenario zu navigieren, gab es bereits 1989 in Pomerleau's Arbeit, die man auf diesem Gebiet als Pionierarbeit verstehen kann[8]. Das Netzwerk Autonomous Land Vehicle In a Neural Network (ALVINN) sollte das NAVLAB steuern, ein Testfahrzeug für Autonome Navigation der Carnegie Mellon University. In 2.2a lässt sich die Architektur nachvollziehen. Der rein visuelle Input (die Blautufenintensität eines Pixels bestimmt das Aktivierungslevel des Inputneurons) wird unterstützt durch eine laserbasierte Abstandsmessung und ein Inputneuron für die Kodierung der „Straßenintensität“, also ob die Straße heller oder dunkler wird. Aus heutiger Sicht ist das Netz mit nur einer hidden Layer mit 29 Neuronen sehr klein, die im weiteren angesprochenen Architekturen haben deutlich mehr Layer und mehrere Hunderttausend Parameter. Zudem interpretiert ALVINN die Aufgabe des Spurfolgens nicht als Regressionsproblem, sondern als Klassifikation. Die Ausgangsneuronen sind eine lineare Repräsentation der Lenkrichtung, die das Fahrzeug in Richtung Fahrbahnmitte steuert. Neuronen in der Mitte stehen für eine Fahrt geradeaus, Neuronen links und rechts für die jeweilige Fahrtrichtung. Grob gesagt gibt das Neuron mit dem höchsten Aktivierungslevel die Fahrtrichtung (den einzuschlagenden Lenkwinkel) an. Im Ergebnis konnte das Netz nach 40 Epochen Training auf simulierten Fahrbahnbildern, zu sehen in 2.2b, einen 400 Meter Weg durch einen Wald mit  $\frac{1}{2}$  m/s sicher abfahren.

#### 2.2.2 NVIDIA DAVE-2

Forschungserkenntnisse der folgenden Jahre trieben die technische Entwicklung voran, auch im Bereich der autonomen Navigation tat sich viel. Zur Orientierung wird eine aktuellere Forschungsarbeit vorgestellt, die einige Aufmerksamkeit auf sich zog. Im Jahr 2016 veröffentlicht das Technologieunternehmen NVIDIA einen eigenen Ansatz [1], basierend auf Versuchen mit dem „DARPA Autonomous Vehicle“ (DAVE) [11] wird dieser „Dave-2“ genannt.// Daten werden hier durch Fahrten auf echten Straßen gesammelt,

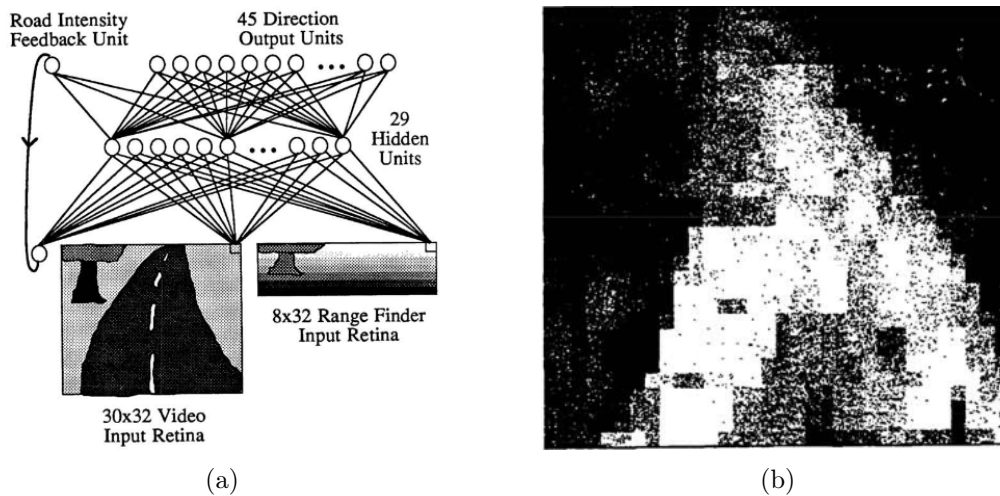


Abbildung 2.2: ALVINN Architektur (a) und simulierte Fahrbahn (b)

wofür drei Kameras in der Windschutzscheibe eines Autos angebracht und Steuerungsdaten über den CAN-Bus des Fahrzeuges ausgelesen werden. Mit diesen Daten wird ein CNN trainiert (2.3), was dann an einer Straßen-Simulation getestet wird. Hervorzuheben ist hier besonders die Verwendung von Convolutional Neural Networks (CNN) und die, im Gegensatz zum bereits erwähnten Ansatz 27 Jahre zuvor, stark gesteigerte Rechenleistung. Folglich können nicht nur Bilder besserer Qualität verarbeitet werden, die Netzarchitektur mit 9 Layern und 250.00 Parametern wäre 1989 nicht in annehmbarer Zeit trainierbar gewesen. Außerdem stellt sich NVIDIA dem Anspruch, eine neuronale Steuerung für öffentliche Straßen zu entwerfen, nicht nur für ein sehr begrenztes Testszenario.

Abbildung 2.3 zeigt die Komponenten des Trainingsprozesses.

Mit dem so trainierten CNN wurden sehr gute Ergebnisse erzielt. Sowohl in der Simulation, als auch auf Testfahrten in realen Straßensituationen, steuerte das neuronale Netz ein Auto nahezu fehlerfrei. Einige der Ansätze und Erkenntnisse aus dieser Arbeit lasse ich für den Entwurf und die Bewertung meiner Arbeit mit einfließen.

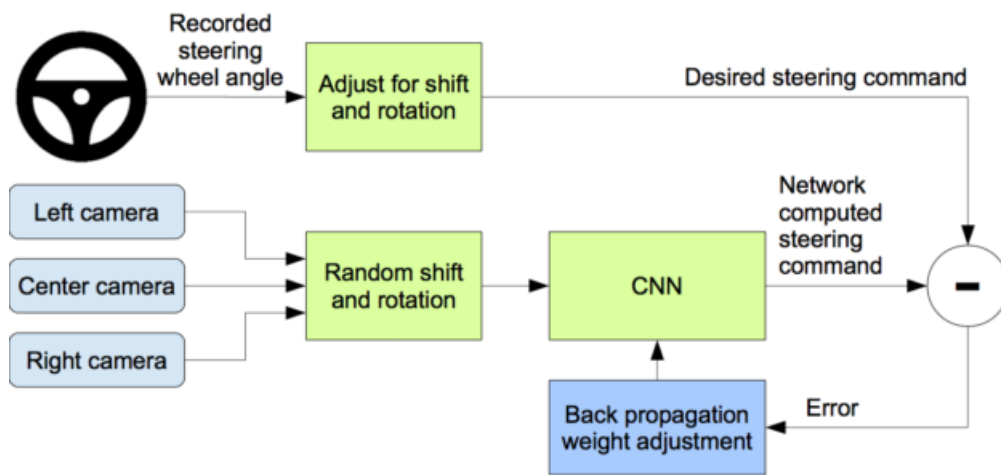


Abbildung 2.3: Komponenten des Trainings  
Quelle: [12]



## 3 Idee

### 3.1 DroNet und Carolo-Cup

Die ETH Zürich entwickelte 2018 eine eigene Architektur, mit dem Ziel durch Training auf Fahrbahnbildern eine Drone zu steuern [7]. Das daraus entstandene, von Aufbau und Größe relativ einfach gehaltene Neuronale Netz, war der Anstoß für diese Arbeit.

Das Netz, siehe Abbildung 3.1, bekommt als Input ein 200x200 Pixel großes Bild in Graustufen, der Output ist ein Lenkwinkel und zusätzlich eine Kollisionswahrscheinlichkeit.

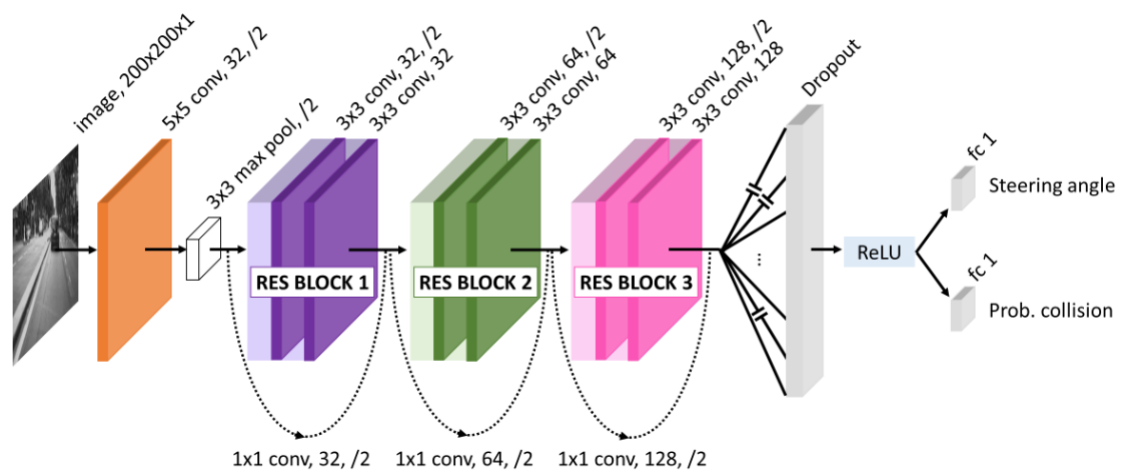


Abbildung 3.1: Architektur DRONET

Trainiert wurde das Netz auf frei verfügbaren Datensätzen der Firma Udacity, bestehend aus Bildern aufgenommen mit Kameras hinter der Windschutzscheibe eines Autos bei stundenlangen Fahrten über Amerikanische Highways. Die Aufnahmen sind mit Fahrdaten verbunden, Zeitstempel, GPS-Daten, Beschleunigungswerte und Lenkwinkel wurden für jedes Bild der Aufnahmen gespeichert. Für das Training von DRONET werden nur die

Bilder der Mittelkamera und der jeweilige Lenkwinkel genutzt.

Zusätzlich hat das Team der ETH Zürich eigene Aufnahmen mithilfe einer am einem Fahrrad montierten Kamera im Straßenverkehr gemacht und diese Aufnahmen manuell mit einer Kollisionswahrscheinlichkeit versehen. Wie bereits erwähnt hat das Netzwerk dementsprechend zwei verschiedene Outputs.

Für diese Arbeit ist aber nur der Lenkwinkel von Interesse, Kollision spielt als Szenario keine Rolle. Im Entwurf werden dementsprechend Anpassungen gemacht.

Es stellte sich heraus, dass das Modell hervorragend generalisierte und eine Drohne sicher durch ein Straßenszenario steuern konnte, wobei das Szenario sich deutlich von den gelernten Unterschied. Diese Eigenschaft von DRONET möchte ich mir im folgenden zu Nutze machen und auf dieser Basis ein Steuerungsmodell für ein RC-Fahrzeug entwickeln.

Die HAW Hamburg nimmt bereits seit einigen Jahren am „Carolo-Cup“ teil, einem Wettbewerb der Technischen Universität Braunschweig. Hier treten Teams einiger deutscher Hochschulen mit RC-Fahrzeugen (Maßstab 1:10) in verschiedenen Disziplinen des autonomen Fahrens gegeneinander an. Der Wettbewerb findet jährlich in Braunschweig auf einem vorbereiteten Kurs statt. Eine hauptsächlich von den HAW Studenten Nils Schönherr und Gunnar Wolfram aufgebaute Plattform, zu sehen in Abbildung 4.1, dient dieser Arbeit als Testplattform. Zum entwickeln der Fahrzeuge steht an der HAW eine Teststrecke zur Verfügung, verschiedene Fahrzeugplattformen sind in der Entwicklung. Das Ziel dieser Bachelorarbeit ist, an einem konkreten Anwendungsfall zu zeigen, dass ein Fahrzeug autonom einen Streckenkurs abfahren kann, indem ein neuronales Netz live Bilder der Strecke auswertet und Lenkinformationen für das Fahrzeug berechnet, schematisch dargestellt in Abbildung (3.2). Da aktive Teams der HAW im Carolo-Cup aktuell klassische (Bildbasierte-) Navigationsansätze verfolgen (Kapitel 1.1), soll die Arbeit auch zeigen, wie eine nächste Generation der Fahralgorithmen für den Carolo-Cup aussehen kann.

Die autonome Fahrleistung auf der Teststrecke ist das Hauptaugenmerk, sie zu messen ist zu analysieren ist Bestandteil der Arbeit.



Abbildung 3.2: Einfacher schematischer Aufbau

## 4 Entwurf

In diesem Kapitel wird der vollständige Entwurf einer Steuerung für ein RC Fahrzeug dargestellt.

Für dieses Kapitel war besonders Buch des Erfinders von Keras, Francois Chollet, hilfreich [2].

Wie im vorangegangenen Kapitel erläutert, entstand die Idee für die Steuerung eines RC-Fahrzeuges mit Neuronalen Netzen im Kontext einer Arbeit der ETH Zürich. Das dort entworfene DroNet wurde bereits kurz vorgestellt und wird die Basis für die neuronale Steuerung sein. Zunächst sollen darum die technischen Aspekte von DroNet hervorgehoben werden:

DroNet ist ein 8-Layer Convolutional Neural Network, das insbesondere aus drei Residual-Blöcken besteht und für den Input eines 200x200 Pixel Bildes in Graustufen zwei Outputs produziert, einmal einen Lenkwinkel aus dem Intervall  $[-1, 1]$  (hierbei gilt; Werte  $< 0$  entsprechen einer Rechtskurve, Werte  $> 0$  einer Linkskurve) und einmal die Kollisionswahrscheinlichkeit in Prozent (als Klassifizierungsproblem in Prozentschritten, nicht kontinuierlich).

DroNet hat  $3.2 \times 10^5$  Parameter und schafft eine Verarbeitungsrate von 20 fps ( Die Drohne, auf der DroNet getestet wurde lieferte Bilder mit 30 Hz). Wie im vorigen Kapitel bereits erwähnt, wurde DroNet auf einem frei verfügbaren Datensatz trainiert.

Im Paper werden die Ergebnisse mit anderen Netzwerken verglichen, die mit dem selben Datensatz trainiert wurden. Anhand verschiedener Metriken zur Messung der Bestimmungsgenauigkeit, wird die Performance auf dem Datensatz bestimmt. Ein solcher Vergleich ist für diese Arbeit jedoch nicht sinnvoll, da es um die Anwendung auf ein spezifisches, reales Szenario geht.

### 4.1 Hardware und Strecke

**Fahrzeug** Das vom HAW-Team aufgebaute Fahrzeug für den Carolo-Cup (Abbildung 4.1 zeigt die Seitenansicht), besteht, abgesehen von Chassis, Motorelektronik und Servos, im Kern aus einem Intel Next Unit of Computing (NUC), auf dem die vollständige Bildverarbeitung und Logik berechnet wird und der Kamera, die die Bilder liefert (montiert an einem Alustab). Die Kamera ist eine UYEYE Schwarzweißkamera der Firma Imaging Development Systems GmbH (IDS), die über USB mit dem NUC verbunden ist. Die Kontrolle per Fernsteuerung, zum Eingreifen im Fehlerfall und zum Positionieren des Fahrzeugs kommuniziert über Funk direkt mit dem Motorcontroller.

Auf dem Intel NUC ist ein Unix Betriebssystem eingerichtet. Eine funktionsfähige Hardware Abstraktion für den Motor und die Steuerungsservos ist vom Carolo-Cup Team bereits entwickelt und steht mir im Weiteren zur Verfügung, diese ist in C++ implementiert.

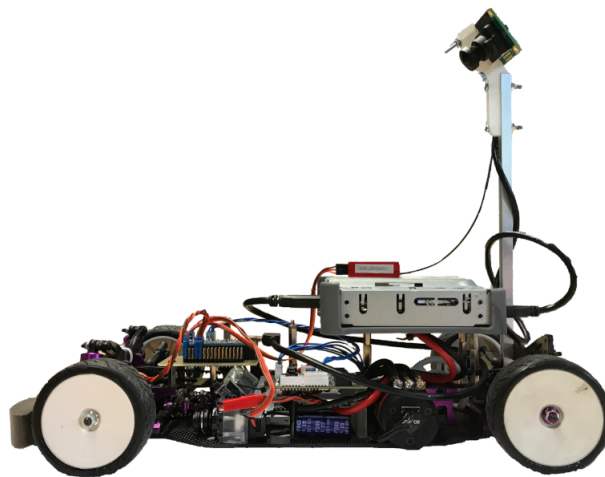


Abbildung 4.1: Das Carolo-Cup Fahrzeug

**Strecke** Die Teststrecke, die an der HAW zur Verfügung steht, ist ein Rundkurs mit verschiedenen Kurven und einem Kreuzungsszenario. Die Länge der äußeren Fahrbahn beträgt 36,1 Meter, die innere Fahrbahn ist 31,3 Meter lang.

Die Fahrbahn mit weißen Band auf schwarzem Untergrund abgeklebt und entspricht der Erscheinung einer Straße, mit unterbrochener Mittellinie. Einige Besonderheiten der

Strecke, wie zum Beispiel Parklücken, sind Teil der Aufgaben des Carolo-Cups, spielen in dieser Arbeit aber keine Rolle. Die Abbildung 4.2 zeigt einen Ausschnitt der Fahrbahn, zu sehen ist eine fast kreisförmige Kurve und die Kreuzungssituation. Der Teststreckenraum ist fensterlos, es wurde sowohl bei Aufnahme der Trainingsbilder, als auch bei den Testfahrten darauf geachtet, dass die volle Deckenbeleuchtung an ist, um für optimale und gleichbleibende Beleuchtungsverhältnisse zu sorgen.

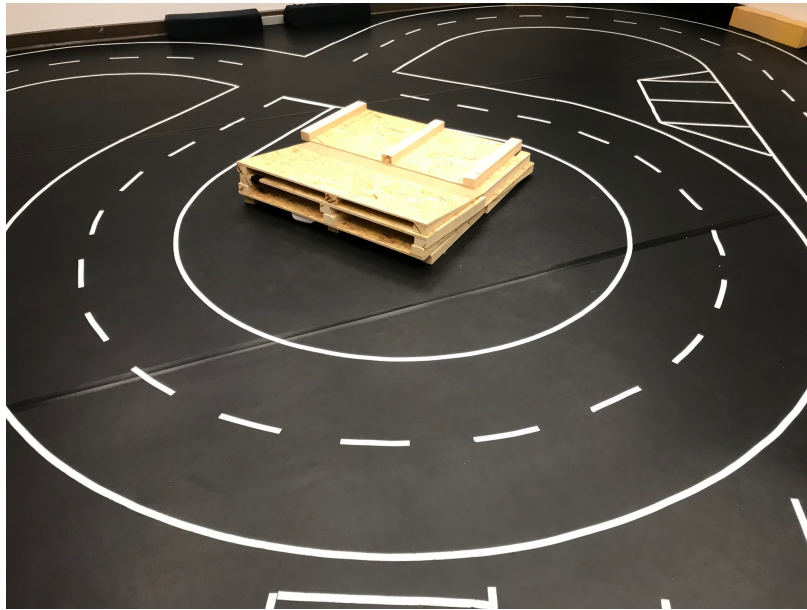


Abbildung 4.2: Ein Ausschnitt der Teststrecke

**Rechnerhardware** Bildverarbeitung und Training des Netzes werden auf einem Windows Computer (Windows 10) mit einer GeForce GTX 950 Grafikkarte berechnet. Da mit einer relativ kleinen Bildanzahl gearbeitet wird und das Neuronale Netz sehr kompakt in Hinblick auf die Parameteranzahl ist, wird keine High-end Grafikkarte benötigt.

## 4.2 Trainingsdaten

Um Trainingsdaten, also Bilder der Teststrecke mit dazugehörigem Lenkwinkel zu bekommen, wird auf einen Algorithmus eines Carolo-Cup Teams (TEAMWORSTCASE) der HAW zurückgegriffen. Trainingsdaten per Hand zu generieren erwies sich als sehr schwierig, per Fernsteuerung lässt sich kaum sauber und kontinuierlich durch den Rundkurs

steuern.

Auf mehreren Fahrten über den Rundkurs mit dem Algorithmus von TEAMWORSTCASE, wurden über 20.000 Bilder gesammelt. Da die Steuerung nicht völlig sauber funktionierte und die Bildfolgen mehrfach Ausreisser enthielten, mussten Bilder per Hand aussortiert werden. Am Ende dieser Vorauswahl standen etwas über 6.000 Bilder für das Training zur Verfügung. Die Auflösung der Aufnahmen ist 752x480 Pixel. Die untere Hälfte der Bilder ist durch eine kamerainterne Vorverarbeitung bereits geschwärzt, TEAMWORSTCASE hat für ihre weitere Verarbeitung so direkt den Teil des Bildes ausgeblendet, auf dem das Fahrzeug selbst zu sehen ist. Da dieser Teil des Bildes im weiteren ohnehin weggeschnitten wird, hat das keine Auswirkungen.

Ein Beispiel aus diesen Fahrbahnaufnahmen zeigt Abbildung 4.3, die Aufnahme ist automatisch in Graustufen, da die Kamera nur in diesem Farbmodus aufnimmt. Diese Rohdaten sind die Basis für das Training und müssen dazu einer Vorverarbeitung unterzogen werden.



Abbildung 4.3: Kameraaufnahme der Fahrbahn

## 4.3 Vorverarbeitung und Fine Tuning

### 4.3.1 Software

Die Bildverarbeitung und das Training sind mit Python programmiert. Für Operationen auf Bildern wird die OpenCV Bibliothek verwendet. Das Neuronale Netz und die Struktur

für Verarbeitung der Bilder und Training sind mit Keras implementiert.

Einzelne Funktionen zur Auswertung und für das Training sind aus dem Repository der DroNet Gruppe der ETH Zürich übernommen, diese sind im Code entsprechend kenntlich gemacht.

Die Abstraktion für den Motorcontroller auf dem Fahrzeug ist in C++ implementiert, die Steuerungskontrolle auf Fahrzeugseite ist daher in C entwickelt.

Die Python Module laufen auf einem Windows Computer in einer virtuellen Umgebung, die mit Anaconda verwaltet wird. Es sind 2 Umgebungen mit Python 3.6.5 eingerichtet. Auf dem Fahrzeug (NUC) kommt zusätzlich Virtual Studio Code (VSCode) zum Einsatz, so wie eine Debugging Umgebung mit PyCharm für die Python Module

Keras wird in Version 2.1.6 auf einem Tensorflow-Backend in Version 1.10.0 verwendet.

### 4.3.2 Preprocessing

Um die Bilder für das Training aufzubereiten, wird eine Verarbeitungspipeline eingerichtet, die in der gleichen Form auch für die Vorverarbeitung im Steuerungsmodus genutzt wird. Die Bilder, die das Netz zum Training zu „sehen“ bekommt, sollen die gleiche Verarbeitung durchlaufen wie die Live-Bilder, die später zum Steuern benutzt werden.

**Lenkwinkelskalierung** Zunächst wird die Verteilung der Lenkwinkel betrachtet. Allerdings muss hier noch eine Skalierung erfolgen, denn die im Carolo-Cup-Fahrzeug verwendete Kodierung entspricht nicht der Kodierung, in der das neuronale Netz trainiert wurde.

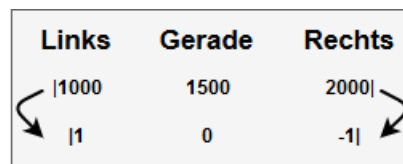


Abbildung 4.4: Skalierung der Lenkwinkel-Wertebereiche

Wie Abbildung 4.4 zeigt, müssen die Wertebereiche umgerechnet werden. Für jeden Lenkwinkel wird zunächst mit  $(Wert - 1500)/500$  auf den Bereich  $[0, 1]$  skaliert, dann werden alle  $Werte \neq 0.0$  mit  $-1.0$  multipliziert. Damit erhält man Werte im Intervall  $[-1, 1]$ , wobei natürlich die Interpretation der negativen Werte als Rechtslenkung und der positiven Werte als Linkslenkung beachtet werden muss.



**Bilderset** In Abbildung 4.5 ist die Lenkwinkelverteilung grafisch aufbereitet. Die x-Achse bildet den Lenkwinkel aus  $[-1, 1]$  ab, die y-Achse die Anzahl der Trainingsbilder mit dem jeweiligen Lenkwinkel. Grafik 4.5a zeigt die Verteilung der Lenkwinkel (zu entsprechenden Bildern) in der gesamten Trainingsbildermenge. Es wird sofort ersichtlich, dass die Mehrheit der gesammelten Lenkdaten von einer Fahrt geradeaus, mit Lenkwinkel um 0.0, stammen. Um eine möglichst ausgeglichene Verteilung zu erreichen wird eine Anpassung vorgenommen.

Alle Bilder, mit Ausnahme derer, die mit einem Lenkwinkel von 0.0 assoziiert sind, werden gespiegelt. So kann die Trainingsdatenmenge verdoppelt werden und gleichzeitig eine ausgeglichene Verteilung der Lenkwinkel erreicht werden. Grafik 4.5b zeigt die Verteilung nach der Anpassung.

Die Spiegelung sorgt automatisch dafür, dass Aufnahmen der rechten Fahrbahnseite zu Aufnahmen der Linken werden. Diese Erweiterung der Trainingsdaten auf beide Spuren wird aber nicht als Problem betrachtet, die Annahme ist, dass Fahrbahneigenschaften unabhängig der Fahrbahnseite gelernt werden können und das Training dadurch robuster wird. So entsteht ein Trainingsbilder-Set von gut 11.000 Bildern.

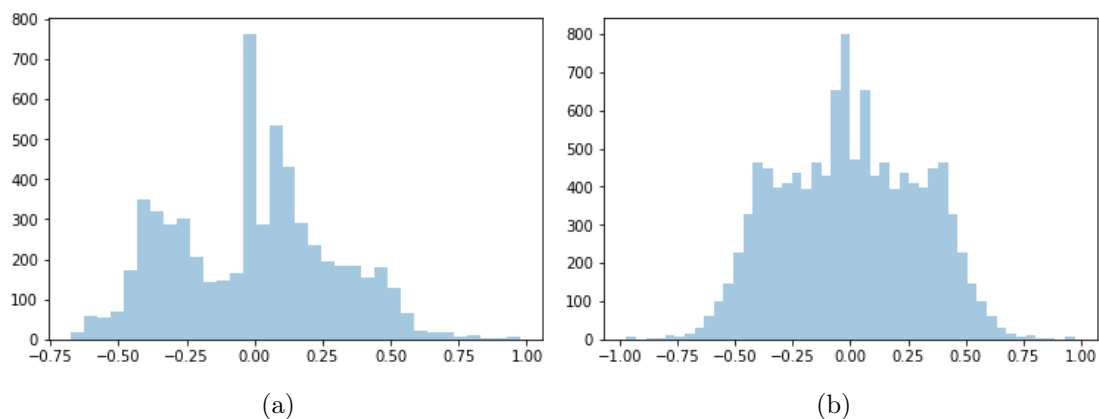


Abbildung 4.5: Lenkwinkelverteilung im Bilderset

Zu jedem Bild muss der dazugehörige Lenkwinkel als Label dem Lernprozess zur Verfügung gestellt werden. Beim Sammeln der Daten sind alle Parameter (Geschwindigkeit, Lenkwinkel, Framenummer) in der Bildbezeichnung hinterlegt worden. Mit einer Parsing Funktion wird die Bezeichnung gefiltert, die Lenkwinkel werden zusammen mit der Framenummer als eindeutigen Identifikator in einer Textdatei gespeichert. Die Textdatei wird für jeden Bildordner, Training-, Validierung- und Testordner angelegt. Die Lenkdaten stehen somit direkt, auch für eine Analyse wie der oben erstellten Verteilung, zur Verfügung.

**Bildverarbeitung** Mit den so vorbereiteten Daten wird im Weiteren eine Pipeline mit Bildoperationen eingerichtet. Die Abbildung 4.6 macht die Struktur der Verarbeitung deutlich.

Als Bildquellen kommen entweder die ueye-Kamera auf dem Fahrzeug in Frage, oder die Trainingsdaten. Im einem ersten Schritt werden die Bilder skaliert und dann auf die Bildgröße 200x200 Bildpunkte zugeschnitten. Hier wurde speziell darauf geachtet, den Bildausschnitt mit dem größten Informationsgehalt zu erhalten, also möglichst viel der Fahrbahn im zugeschnittenen Bild.

Die Aufnahmen sind recht dunkel, selbst bei voll eingeschaltetem Raumlicht waren keine besseren Aufnahmen zu bekommen. Hier wird ein Histogrammausgleich für jedes Bild angewandt. Die Verteilung der Grauwerte wird so besser auf den gesamten Wertebereich aufgezoogen, die Bilder werden heller.

Anschließend werden die Grauwerte auf den Wertebereich  $[0, 1]$  normalisiert. Für die Berechnung in CNNs ein üblicher Schritt, die Produkte der Multiplikationen bleiben in einem kleinen Wertebereich und ermöglichen so ein effizienteres Training. Außerdem wird noch eine Umformung des Bildarrays von einem zweidimensionalen zu einem dreidimensionalen vorgenommen, dies ist im weiteren ebenfalls für die Verarbeitung im Netz von Bedeutung, auch wenn hier nur eine 1 für die eine Farbwertdimension (Grauwert) angefügt wird.

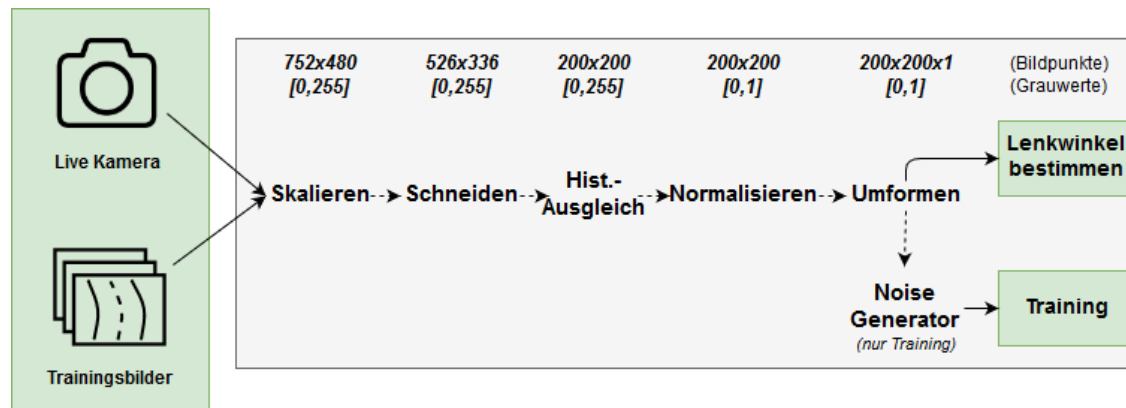


Abbildung 4.6: Bildverarbeitungsschritte

Für das Training wird auf einem Teil der Bilder ein Filter angewendet, der das Bild optisch rauschen lässt, um das Lernen robuster zu machen. Hierzu werden zufällige Werte aus einer (Gausschen-)Normalverteilung mit Mittelpunkt 0 gezogen und auf die Pixelwerte addiert. Für positive Werte werden die Pixel somit heller, für negative dunkler. So wird ein körniges, unscharfes Erscheinen des Bildes erreicht.

Darstellung 4.7 zeigt Auszüge aus den Bearbeitungsschritten, nach dem Zuschneiden, nach dem Histogrammausgleich und nach Durchlaufen eines Rausch-Filters.

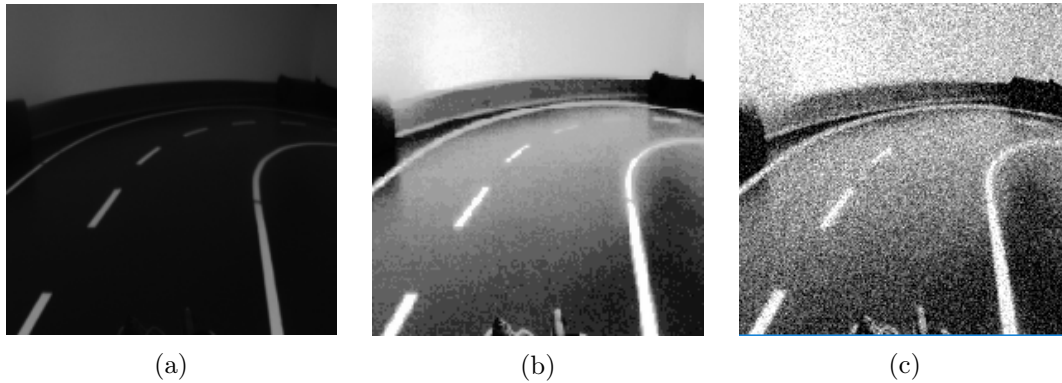


Abbildung 4.7: Beispiele verschiedener Bearbeitungsstufen, zugeschnitten (a), nach Histogrammausgleich (b) und nach Rausch-Filter (c)

Das Bildersetz enthält jetzt gut 11.000 Bilder und muss für die Verwendung im Trainingsprozess in Trainings- und Validierungsbilder aufgeteilt werden. Die hier verwendete Aufteilung ist im Verhältnis 1 zu 4. Damit entsteht ein Trainingsbildersetz mit 8.800 Bildern und ein Validierungsbildersetz mit 2.200 Bildern.

### 4.3.3 Fine Tuning

**Netzarchitektur** Das CNN DroNet, welches als Grundlage verwendet wird, ist bereits mit Bildern von Fahrbahnen aus dem regulären Straßenverkehr trainiert. Eigenschaften einer Fahrbahn sind also, so die Annahme, bereits in der Gewichtsverteilung der einzelnen Layer repräsentiert. Die für diese Arbeit interessante Strecke fällt natürlich auch in die Kategorie Fahrbahn, allerdings unterscheiden sich die Aufnahmen der Teststrecke von denen, einer echten Fahrbahn. Es ist keine Bebauung vorhanden, ebenso gibt es keine anderen Verkehrsteilnehmer und die Art der Straßenführung unterscheidet sich von realen Situationen.

Statt die Gewichtsmatrizen des gesamten Netzwerkes anzupassen, wird nur ein Fine-Tuning des letzten Residual Blocks vorgenommen. Die architektonische Aufteilung in diese Blöcke ermöglicht ein gutes Aufteilen des Netzes in einen Teil mit festgesetzten bzw. eingefrorenen Gewichten und den Teil, der trainiert wird.

Abbildung 4.8 veranschaulicht diese Aufteilung. Der Teil des Netzes in roten Klammern wird vom Training ausgeschlossen, der grün eingeklammerte Block wird trainiert. Die

Annahme ist, dass die ersten beiden Blöcke bereits allgemeine Eigenschaften einer Fahrbahn erlernt haben, die gut auf neue Szenarien generalisieren. Der letzte Residual Block enthält knapp 70% der Parameter des gesamten Netzes, die Annahme ist, dass hier die allgemeinen Eigenschaften zu einer komplexeren Repräsentation zusammenwachsen, die gut auf die Testfahrbahn generalisiert. Darüberhinaus soll vermieden werden, dass das recht kleine Bildersetz mit insgesamt 11.000 Bildern zu einer Überanpassung (Overfitting) führt. Das könnte auftreten, da die Aufnahmen auf Fahrsituationen der Teststrecke begrenzt sind, die keine große Heterogenität aufweisen. Nur einen Teil der Gewichte des Netzes zu verändern, erscheint auch in diesem Kontext sinnvoll.

Zusätzlich wird die Ausgabe des Netzes verändert. Die Kollisionswahrscheinlichkeit spielt für diese Arbeit keine Rolle und die Trainingsdaten enthalten keine entsprechenden Label, daher wird das entsprechende Ausgabe-Layer entfernt. Das Feedback für die Berechnung der Gewichtsadjustierungen kommt nur vom Lenkwinkel.

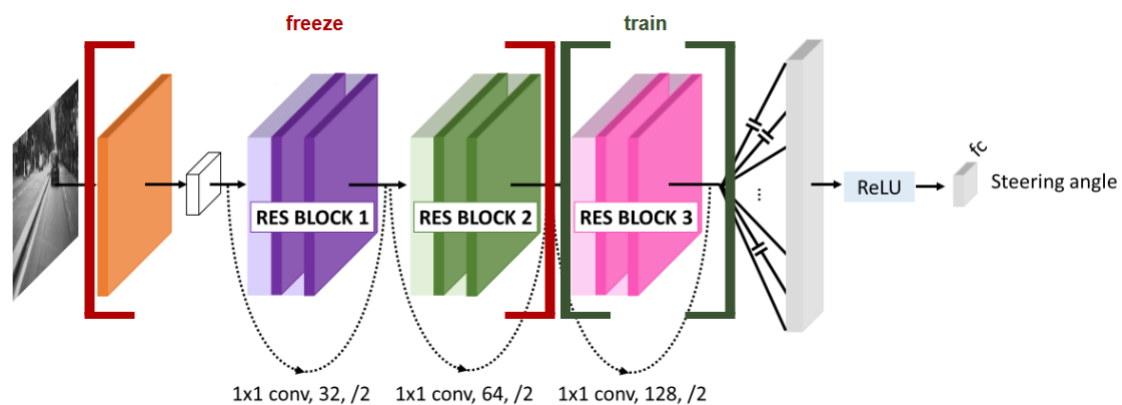


Abbildung 4.8: Angepasste Architektur

**Lernarchitektur** Im Folgenden wird die Konfiguration für das Fine-Tuning vorgenommen und die Komponentenstruktur des Lernverfahrens erläutert.

Vor dem Fine-Tuning muss der Trainingsprozess konfiguriert werden, dafür werden vor dem Beginn die Hyperparameter festgelegt.

Das Adam Optimierungsverfahren [5] wird als Optimierungsalgorithmus aus dem Dronet Training übernommen. Die Lernrate wird zu Beginn auf 0.0001 festgelegt mit einem Verfall (Decay) von  $10 \times 10^{-5}$ . Es werden 100 Trainingsepochen festgelegt und es wird in Gruppen (Batches) von jeweils 32 Bildern trainiert. Für das Training wird außerdem eine

Shuffle-Funktion aktiviert, die Bilder werden so in einer zufälligen Reihenfolge trainiert. Einen Überblick über das Lernverfahren zeigt Abbildung 4.9. Die Bilder durchlaufen im ersten Schritt die Preprocessing-Komponente, die Lenkwinkel werden ausgelesen und skaliert, wie bereits im Abschnitt zur Vorverarbeitung beschrieben. Das CNN generiert für ein Bild einen Lenkwinkel, welcher dann mit dem tatsächlichen (gewünschten) Lenkwinkel, der diesem Bild zugeordnet ist, verglichen wird. Der Unterschied zwischen den Lenkwerten, der Fehler, wird dem CNN als Feedback wieder zurückgegeben. Die Bestimmung des Fehlerwerts erfolgt immer für eine Batch, also eine Gruppe von *32 Bildern*.

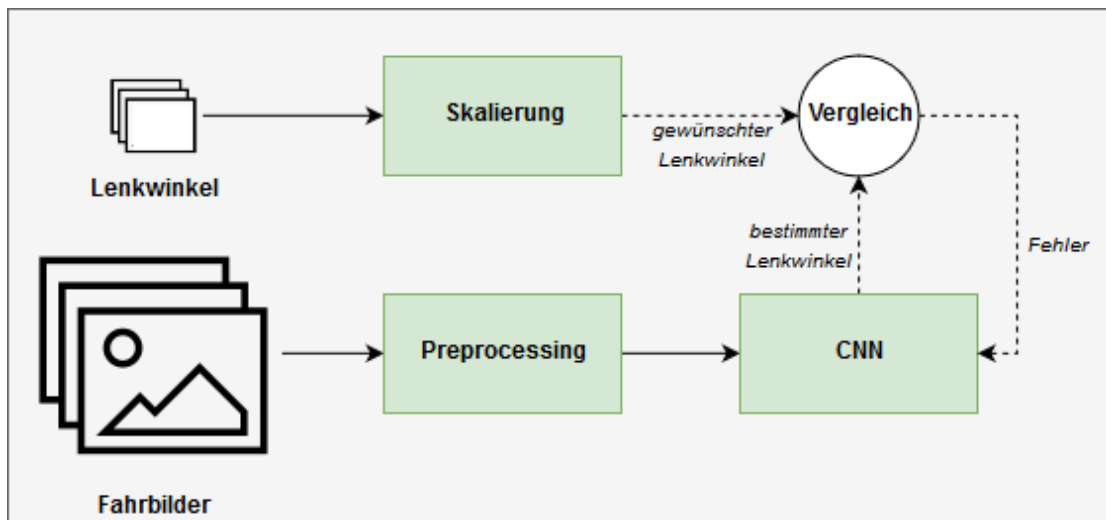


Abbildung 4.9: Struktur des Lernprozesses

### 4.4 Steuerung

Um aus der Hardware, also der Fahrzeugplattform und der Software, dem neuronalen Netz, eine Steuerungseinheit zu machen, müssen beide Teile verbunden werden. Hierbei ist besonders die Kommunikation wichtig, die Lenkdaten müssen von der erzeugenden Komponente (CNN) zur ausführenden Komponente (Hardwareabstraktion) übertragen werden. Die Steuerungsarchitektur ist in Abbildung 4.10 zu sehen.

Über eine Python Schnittstelle zur ueye-Kamera wird die Kamera konfiguriert und Bilddaten ausgelesen. Nach der bereits besprochenen Vorverarbeitung (Preprocessing) werden die Daten an das neuronale Netz (CNN) weitergereicht.

Der hier bestimmte Lenkwinkel wird nun zur in C/C++ programmierten Steuerung weitergegeben. Dazu werden Unix Domain Sockets (UDS) verwendet. Die Kommunikation erfolgt statt über eine IP-Adresse über einen File Deskriptor im Unix Betriebssystem. Ein Stream-Protokoll (entspricht TCP) nutzt den File-Deskriptor als Kommunikationsendpunkt für die Interprozesskommunikation. Der UDS-Client auf der Python-Seite schickt die Lenkdaten direkt an einen UDS-Server auf der C/C++ Seite. Dabei werden die Daten als UTF-8 kodierte Bytestreams verschickt. Die so erhaltenen Daten werden vom UDS-Server entpackt.

Für die direkte Ansteuerung des Lenkungsservos wird der erhaltene Wert auf zwei Nachkommastellen gekürzt und dann direkt an die Hardware weitergegeben. Es erfolgt keine zeitliche Kontrolle auf Zugehörigkeit einzelner Lenkdaten zu Kamerabildern. Die Steuerung soll möglichst hohen Durchsatz haben um möglichst viele Frames in einer Sekunde zur Lenkwinkelermittlung nutzen zu können, daher werden immer die aktuellsten erhaltenen Lenkdaten genutzt. Wird ein Datum nicht übertragen, ist es aufgrund der dynamischen Fahrsituation ohnehin schon weniger als eine Sekunde später nicht mehr zur Lenkung zu gebrauchen.

Die so gebaute Steuerungskomponente kann zum Fahren des Fahrzeuges verwendet werden. Die Komponenten werden über eine ssh-Verbindung gestartet, die Kontrolle für den autonomen Modus wird dem Fahrzeug über die Funkfernbedienung übergeben. So ist auch ein Eingriff per Hand möglich.

Die Qualität der Steuerung und das Fahrverhalten ist im Weiteren Gegenstand einer Analyse, in der auch aufgezeigt wird, welche Fahrbahn-Eigenschaften das neuronale Netz interpretiert.

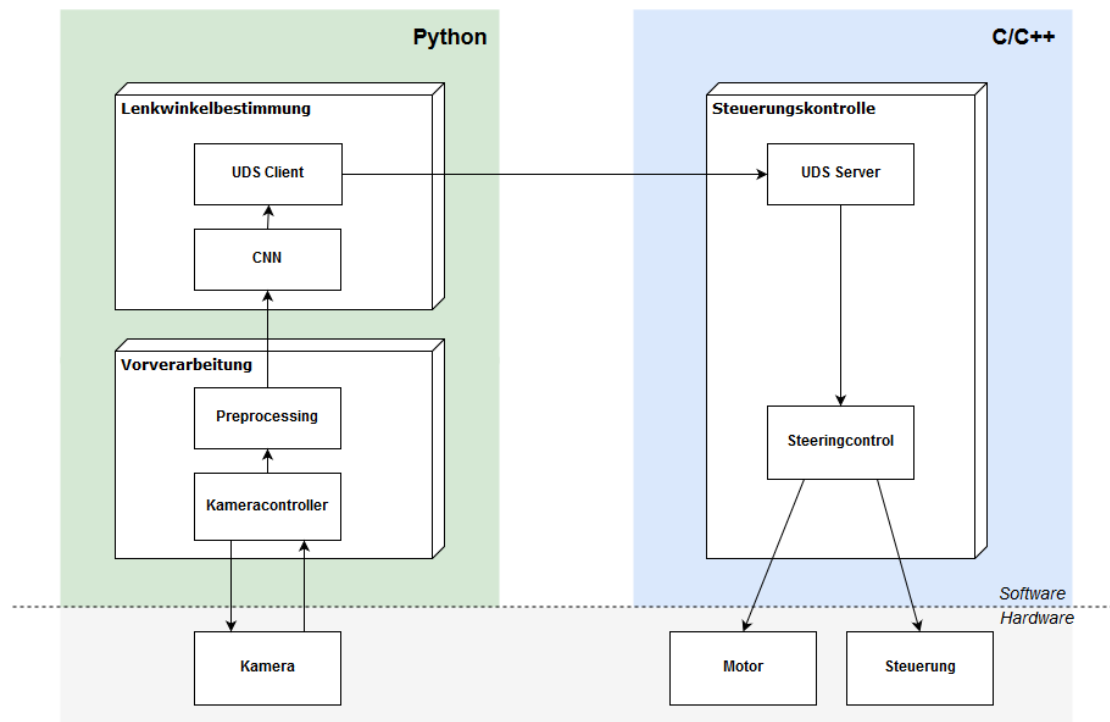


Abbildung 4.10: Komponentenstruktur und Verbindung der Neuronalen Steuerung

## 5 Auswertung und Szenarien

In diesem Kapitel wird der Trainingsprozess analysiert und kritisch bewertet, zudem wird das entworfene neuronale Steuerungssystem einem Test unterzogen. Verschiedene Fahrszenarien werden untersucht, zum Schluss bietet eine grafische Auswertung einen Einblick in die Perspektive aus Sicht des neuronalen Netzes.

### 5.1 Training

Das Fine-Tuning wurde mit den im vorangegangenen Kapitel festgelegten Konfigurationen durchgeführt. In diesem Abschnitt erfolgt eine Betrachtung des erfolgten Trainings. In Abbildung 5.1 sind Trainingsfehler und Validierungsfehler (y-Achse) über 100 Epochen (x-Achse) dargestellt.

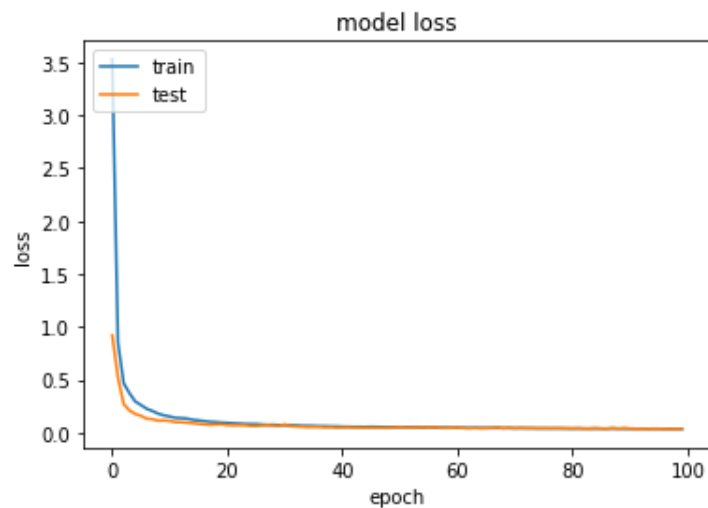


Abbildung 5.1: Trainingsfehler (train) und Validierungsfehler (val) über 100 Epochen

Festzustellen ist, dass beide Fehlerwerte sich zunächst stark fallend 0.0 annähern und zwischen den Epochen 20 – 25 und 100 gleichbleibend 0.0 zu sein scheinen. Aufgrund der



Start-Trainingsfehler von etwa 3.5 lassen sich sehr kleine Werte nicht grafisch beurteilen. Ein direkter Blick auf die Fehlerwerte zeigt, dass beide sich im Bereich  $10 \times 10^{-2}$  weiter 0.0 annähern, mit dem Besten Validierungsfehler von 0.0344 in Epoche 197. Es besteht keine ausgeprägte Divergenz zwischen Trainings- und Validierungsfehler, das ist positiv zu beurteilen.

Bei kritischer Betrachtung muss angemerkt werden, dass die Regressionkurve, im konkreten Fall die Annäherung des Lenkwinkels, sehr rapide fällt. Dieses kann Zeichen eines Overfittings sein. Die Bilder der Teststrecke könnten in der Menge zu homogen sein, die Teststrecke ist im Angesicht der dort aufgenommenen Bildermenge relativ kurz, viele Bilder sind ähnlich mit nur leicht unterschiedlichen Lenkwerten. Aufgrund der räumlichen Abgeschlossenheit der Teststrecke gibt es dazu wenig Variabilität im Bildhintergrund, das könnte die Gleichartigkeit der Bilder verstärken.

Allerdings ist die schnelle Konvergenz nicht zwingend negativ anzumerken, das neuronale Netz war schließlich schon mit derselben Aufgabe auf einer ähnlichen Bildermenge trainiert worden. Dazu kommt, dass nur ein Teil der Layer überhaupt für das Training, also eine Veränderung der Gewichte, freigegeben war. Im Angesicht dieser Umstände, können die Fehlerkurven in Abbildung 5.1 auch für eine schnelle, effiziente Anpassung der Gewichte des letzten Residual-Blocks stehen.

Wie das CNN als Steuerungslogik in realen Szenarien abschneidet, wird in den nächsten Abschnitten untersucht.

### 5.2 Testfahrt

Um die Lenkwinkelbestimmung des Netzes zu Testen, wird zunächst frei auf der Strecke gefahren. Das Fahrzeug startet auf einer Geraden und fährt den Kurs überwiegend fehlerfrei ab. Der Onboard-Rechner (NUC) des Fahrzeugs verarbeitet die Bilder mit  $20fps$ , getestet wird bis zu einer Geschwindigkeit von  $1.2\text{ m/s}$ , ab da nimmt die Lenkfehlerhäufigkeit stark zu. Ein Lenkwinkelupdate (verarbeiteter Frame) pro gefahrene  $6\text{ cm}$  ist zum sauberen Fahren die Grenze.

**Autonomiemetrik** Die Fahrleistung auf der Strecke soll gemessen werden, um Sie direkt mit anderen Algorithmen vergleichen zu können. Dazu wird eine Autonomie-Metrik

bestimmt [1]. Der Autonomiewert wird definiert als

$$\left(1 - \frac{\text{Anzahl Fehler} \cdot 2s}{\text{Fahrzeit in Sekunden}}\right) \cdot 100, \quad (5.1)$$

wobei die Anzahl der Fehler die Summe der Situationen ist, in der das Fahrzeug die Außen- oder Mittellinie mit mehr als zwei Reifen überquert hat. Das Fahrzeug kann sich danach selbst wieder auf die Strecke bringen, oder per Hand zurückgelenkt werden. Die Zeit, die vergeht, bis das Fahrzeug wieder sauber auf die Straße zurückgekehrt ist, wird mit 2 Sekunden angenommen. Die Fahrzeit in Sekunden beschreibt die Dauer des Versuchs.

Für den Test werden zwei Runden auf der Teststrecke gefahren, einmal im Uhrzeigersinn und einmal gegen den Uhrzeigersinn, beide Runden auf der jeweils Rechten Straßenseite. Als Geschwindigkeit wird  $\frac{1}{2}$  m/s festgelegt. In jede Richtung der Strecke wird 60 Sekunden gefahren, das entspricht je einer Runde. Die Gesamtfahrzeit für alle Tests ist damit 120 Sekunden.

Verglichen wird das unveränderte DroNet, die abgestimmte (finegetunte) Variante (bezeichnet als BA-RR) und der Steuerungsalgorithmus (Linienerkennung), der von der diesjährigen Carolo-Cup-Projektgruppe entwickelt wurde. Die erste Runde ist mit, die zweite gegen den Uhrzeigersinn. Die Testfahrten ergaben die in Tabelle 5.1 gesammelten Daten.

Tabelle 5.1: Algorithmen im Vergleich

<b>Algorithmus</b>	<b>Fehler Runde 1</b>	<b>Fehler Runde 2</b>
DroNet	16	12
Carolo-Projekt	7	11
BA-RR	3	5

Der Autonomiewert wird nach der Formel berechnet und liefert die in Tabelle 5.2 vermerkten Werte. Der Autonomiewert ist eine Prozentangabe und ist auf ganze Zahlen gerundet.

Tabelle 5.2: Vergleich der Autonomie

<b>Algorithmus</b>	<b>Autonomiewert</b>
DroNet	53 %
Carolo-Projekt	70 %
BA-RR	87 %

Der ursprüngliche DroNet Algorithmus ist nur 53% der Fahrzeit autonom unterwegs, was nicht überrascht, da er in einem anderen Anwendungsszenario entstand. Der Algorithmus des aktuellen Carolo-Projektteams ist deutlich besser, hat allerdings auch viele Fahrfehler gemacht. Die in dieser Arbeit entwickelte Steuerung (BA-RR) ist 87% der Fahrzeit autonom unterwegs, hier zeigt sich die deutliche Verbesserung, die insbesondere im Gegensatz zum DroNet Algorithmus erreicht wurde. Außerdem wird deutlich, dass eine Steuerung mittels statistischer Auswertung von Bilddaten (CNN) nicht nur möglich ist, sondern einer Steuerung mittels klassischer Linienerkennung sogar überlegen sein kann. Im Folgenden wird das Verhalten des Fahrzeugs in bestimmten Szenarien untersucht.

### 5.3 Einzelszenarien

**Fahrbahn wiederfinden** Im Hinblick auf die Eingangs angesprochene mögliche Overfitting-Problematik, wird hier eine praktische Herangehensweise an die Analyse gemacht. Im Trainingsbildersatz sind keine Bilder enthalten, in denen das Fahrzeug aus einer Fehler-situation wieder auf die Straße navigiert. Genauer gesagt, in denen das Fahrzeug nicht oder nur teilweise auf der Strecke steht und den Weg in die richtige Fahrspur selbst findet. Somit konnten diese Szenarien beim Training auch nicht „auswendig“ gelernt werden. Diese Fehlersituationen werden jetzt künstlich erzeugt und das Fahrverhalten betrachtet. Abbildung 5.2 zeigt beispielhaft zwei solche Szenarien, einmal steht das Fahrzeug teilweise auf der Straße (5.2a), im anderen Beispiel steht es komplett außerhalb der Fahrbahn (5.2b).

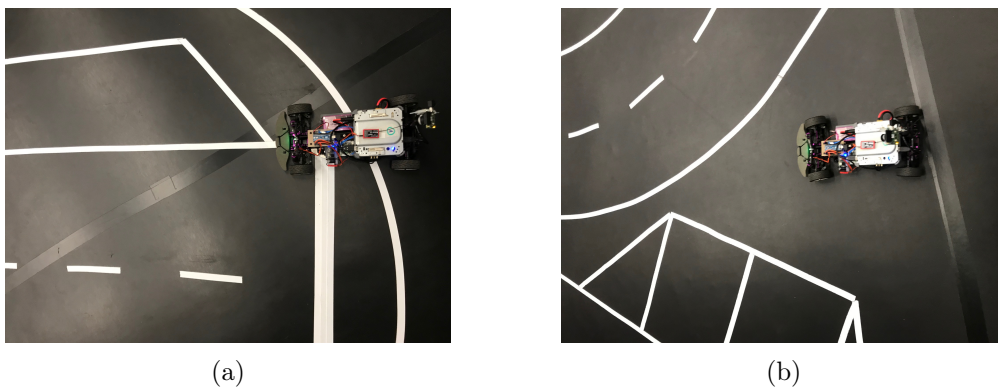


Abbildung 5.2: Fehlerszenarien

Es wurden jeweils 10 verschiedene Szenarien mit dem Fahrzeug teilweise auf der Fahr-

bahn und 10 mit dem Fahrzeug außerhalb der Fahrbahn (jedoch mit Kamerasicht auf zumindest einen Teil der Fahrbahn) getestet. In 10/10 Szenarien findet das Fahrzeug den Weg auf die Fahrspur, wenn es teilweise darauf steht. In 9/10 Fällen findet es die Fahrspur, wenn es vollständig außerhalb der Fahrbahn steht.

Aus diesem Versuch lässt sich die Vermutung ableiten, dass die Orientierung an Fahrbahneigenschaften nicht nur in den trainierten Fällen funktioniert. Die Fahrspur wird auch aus vorher nicht gesehenen Blickwinkeln gefunden.

**Kreuzung** Da die Teststrecke nur eine Kreuzung hat, sind in den Trainingsdaten nur wenige Bilder von Kreuzungsszenarien, wie dem in Abbildung 5.3 gezeigten, vorhanden. Das Verhalten in diesen speziellen Situationen, in denen ein Teil der Fahrbahnmarkierung fehlt, ist besonders interessant.



Abbildung 5.3: Kreuzungsszenario aus Sicht der Fahrzeugkamera

Tatsächlich erweisen sich die Kreuzungssituationen als größte Fehlerquelle. Wird die Kreuzung gerade angefahren, wird sie in 9/10 Versuchen sauber überquert. Ist das Fahrzeug allerdings nicht mittig auf der Fahrbahn, sondern fährt die Kreuzung in einem leicht veränderten Winkel an, wird sie nur noch in 5/10 Versuchen sauber durchfahren. In 50% der Testfälle mit leicht verändertem Blickwinkel verlässt das Fahrzeug die Fahrspur und wechselt die Spur.

Fehlt die Seitenlinie zur Orientierung, erschwert das die Lenkwinkelbestimmung. Durch eine leicht schräge Anfahrt der Kreuzung (zum Beispiel nach einer nicht ganz sauber durchfahrenen Kurve) gerät schnell die andere Fahrbahn in den Blickwinkel, welche dann zur Orientierung verwendet wird. Das Fahrzeug wechselt die Fahrspur.

## 5.4 Saliency

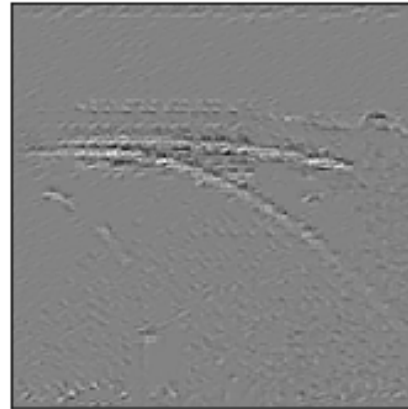
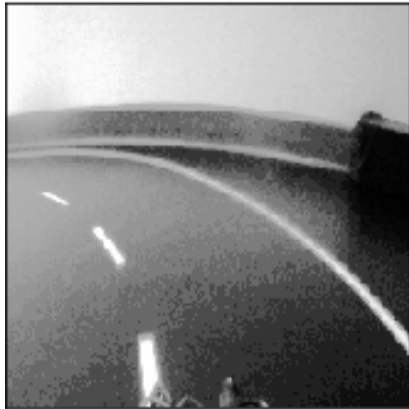
Gerade vor dem Hintergrund der Analyse des Netzes ist es interessant zu Wissen, welche Bereiche des Bildes für die Bestimmung des Lenkwinkels tatsächlich von Bedeutung sind. Als Mensch kennt man das Konzept einer Fahrbahnmarkierung und der daraus entstehenden Fahrspur sehr genau. Man könnte davon ausgehen, dass für ein neuronales Netz diese Bereiche eines Bildes deswegen ebenfalls besonders interessant sind, um die Fahrtrichtung zu bestimmen.

Auch in Bezug auf die in vorangegangenen Kapitel besprochenen Problematiken, liegt ein gesteigertes Interesse vor, die wichtigen Bereiche der Fahrbahnbilder ausfindig zu machen.

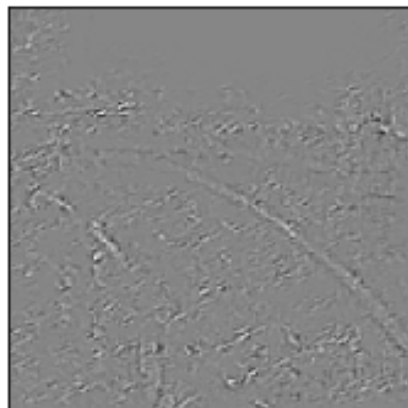
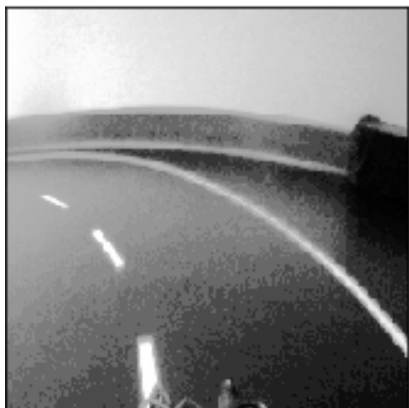
Für die Visualisierung werden so genannte Saliency-Maps verwendet. Diese wurden 2013 in einem Papert [9] erstmals vorgestellt. Die Idee ist, jene Bildpunkte zu identifizieren, welche für die größten Veränderungen im Output sorgen. Für diese Arbeit bedeutet das, genau die Bildpunkte visuell hervorzuheben, welche für eine große Veränderung im Lenkwinkeloutput sorgen. Dafür wird die Veränderung (Gradient) des Outputs (Lenkwinkel) in Bezug auf das Eingabebild berechnet, also das Verhältnis einer Änderung im Output zu einer Änderung im Input:  $\frac{\partial \text{Output}}{\partial \text{Input}}$ . Für diese Analyse wird Keras-vis [14] genutzt, eine Visualisierungs-API für Keras.

Um die Repräsentation der Bildeigenschaften im Netz deutlich zu machen, wird eine Saliency Map für jeden der drei Residual-Blöcke bestimmt. So lässt sich nachvollziehen, wie sich die ausschlaggebenden Bildbereiche auf dem Weg durch das Netz verändern. In Abbildung 5.4 sind diese drei Saliency Maps für ein Beispielbild einer Linkskurve dargestellt.

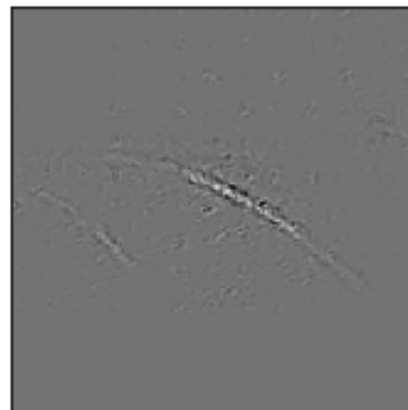
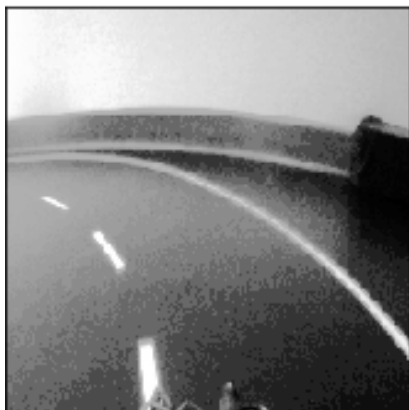
Die hellen Bildbereiche sind eben die, die für eine Erhöhung des Outputs sorgen, also für einen positiven Lenkwinkel. Zur Erinnerung, ein positiver Lenkwinkel codiert eine Linkskurve. Die Saliency-Map wurde für die jeweils letzte Convolutional-Layer des Blocks berechnet.



(a) Residual-Block 1



(b) Residual Block 2



(c) Residual Block 3

Abbildung 5.4: Ausschlaggebende Bildbereiche

## 6 Resümee

Das Ziel dieser Arbeit war, eine bildbasierte neuronale Steuerung für ein RC-Fahrzeug zu entwickeln, was gelungen ist. Durch Anpassung (Fine Tuning) eines bereits trainierten Convolutional Neural Network entstand eine geeignete Steuerung. Fahrten auf einer größeren Strecke konnten mangels Verfügbarkeit noch nicht durchgeführt werden. Zukünftige Testfahrten könnten unter Umständen auf der Teststrecke der TU Braunschweig durchgeführt werden, die ganzjährig aufgebaut ist.

Im Performance-Vergleich auf Testfahrten schneidet das CNN hervorragend ab. Die Teststrecke kann nicht nur fast fehlerfrei durchfahren werden, im direkten Vergleich mit dem Steuerungsalgorithmus des Carolo-Cup Teams des WS18/19 liegt die hier entwickelte neuronale Steuerung vorn. Ein zusätzlicher Vergleich mit dem Ursprungsnetz DroNet zeigt, dass das Fine Tuning die Lenkwinkelbestimmung entscheidend verbessert hat. Einschränkung muss hier angemerkt werden, dass das Fine Tuning eine Anpassung auf einen eingeschränkten, störungsarmen Testbereich gemacht hat.

Zusätzlich wird eine grafische Analyse vorgenommen. Die Salient-Objects, also die für die Bestimmung des Lenkwinkels maßgeblichen Teile des Bilds wurden hervorgehoben. So ließ sich zeigen, dass bei dem CNN was einem Fine-Tuning unterzogen wurde, besonders die Fahrbahnmarkierungen ausschlaggebend waren, was der intuitiven Erwartung entspricht. Ein Vergleich zu DroNet macht außerdem auch grafisch die Vorteile des Fine Tuning deutlich. Die „Aufmerksamkeit“ des Netzes konzentriert sich direkter auf die Fahrbahnmarkierungen, wie **bla**

Darüberhinaus wurden Sonderszenarien betrachtet, wie das Verhalten in Szenarien mit unterbrochener Fahrbahnmarkierung und das Verhalten in Situationen, in denen das Fahrzeug entweder nur teilweise oder gar nicht auf der Fahrbahn startet.(Bilder)

- Teststrecke nicht lang genug -Teststrecke zu sauber, d.h. zu optimal zum lernen (Streckenführung perfekt und sauber)

- Trainingsdaten sammeln optimieren

extra trainings parameter - Hindernisse einbauen -geschwindigkeit einbauen



# Literaturverzeichnis

- [1] BOJARSKI, Mariusz ; DEL TESTA, Davide ; DWORAKOWSKI, Daniel ; FIRNER, Bernhard ; FLEPP, Beat ; GOYAL, Prasoon ; JACKEL, Lawrence D. ; MONFORT, Mathew ; MULLER, Urs ; ZHANG, Jiakai u. a.: End to end learning for self-driving cars. In: *arXiv preprint arXiv:1604.07316* (2016)
- [2] CHOLLET, F.: *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. mitp-Verlag, 2018 (mitp Professional). – URL <https://books.google.de/books?id=ouVcDwAAQBAJ>. – ISBN 9783958458406
- [3] GOODFELLOW, Ian ; BENGIO, Yoshua ; COURVILLE, Aaron: *Deep Learning*. MIT Press, 2016. – <http://www.deeplearningbook.org>
- [4] HE, Kaiming ; ZHANG, Xiangyu ; REN, Shaoqing ; SUN, Jian: Deep Residual Learning for Image Recognition. In: *CoRR* abs/1512.03385 (2015). – URL <http://arxiv.org/abs/1512.03385>
- [5] KINGMA, Diederik P. ; BA, Jimmy: Adam: A method for stochastic optimization. In: *arXiv preprint arXiv:1412.6980* (2014)
- [6] KRIZHEVSKY, Alex ; SUTSKEVER, Ilya ; HINTON, Geoffrey E.: Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*, 2012, S. 1097–1105
- [7] LOQUERCIO, Antonio ; MAQUEDA, Ana I. ; BLANCO, Carlos R. D. ; SCARAMUZZA, Davide: Dronet: Learning to Fly by Driving. In: *IEEE Robotics and Automation Letters* (2018)
- [8] POMERLEAU, Dean A.: Alvin: An autonomous land vehicle in a neural network. In: *Advances in neural information processing systems*, 1989, S. 305–313
- [9] SIMONYAN, Karen ; VEDALDI, Andrea ; ZISSERMAN, Andrew: Deep inside convolutional networks: Visualising image classification models and saliency maps. In: *arXiv preprint arXiv:1312.6034* (2013)

- [10] WALTER, W G.: An imitation of life. In: *Scientific American* 182 (1950), Nr. 5, S. 42–45

## Internetquellen

- [I11] *Net-Scale Technologies Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004.* <http://net-scale.com/doc/net-scale-dave-report.pdf>. – Accessed: 2019-03-08
- [I12] *NVIDIA Komponenten des Steuerungssystem.* <https://devblogs.nvidia.com/wp-content/uploads/2016/08/training-624x291.png>. – Accessed: 2019-03-08
- [I13] *Residual Block Schematischer Aufbau.* [https://cdn-images-1.medium.com/max/1600/1\\*D0F3UitQ2l5Q0Ak-tjEdJg.png](https://cdn-images-1.medium.com/max/1600/1*D0F3UitQ2l5Q0Ak-tjEdJg.png). – Accessed: 2019-03-10
- [I14] KOTIKALAPUDI, Raghavendra ; CONTRIBUTORS: *keras-vis*. <https://github.com/raghakot/keras-vis>. 2017

# Glossar

**Adam** Ein Optimierungsalgorithmus für stochastische Gradientenverfahren.

**ALVINN** Bezeichnung für ein neuronales Netz der Carnegie Mellon Universität.

**Anaconda** Open Source Python Distribution für wissenschaftliches Programmieren.

**Carolo-Cup** Studentischer Wettbewerb der TU Braunschweig, Modellfahrzeuge treten in Disziplinen autonom gegeneinander an.

**DroNet** Bezeichnung für ein neuronales Netz, entwickelt von einer Arbeitsgruppe der ETH Zürich.

**ETH Zürich** Eidgenössische Technische Hochschule Zürich.

**HAW Hamburg** Die HAW Hamburg ist die ehemalige Fachhochschule am Berliner Tor.

**IDS** Hersteller von Industriekameras.

**Intel NUC** Kleiner Barebone-Computer entwickelt von der Firma Intel.

**NAVLAB** Eine Serie autonomer und semi-autonomer Fahrzeuge der Robotik-Gruppe der Carnegie Mellon Universität.

**OpenCV** OpenCV ist eine Bibliothek, die Funktionen für Bilderverarbeitung und Maschinelles Sehen anbietet.

**PyCharm** IDE speziell für Python.

**Udacity** Private Online-Bildungsorganisation gegründet von Sebastian Thrun.

**UDS** Socket Kommunikation mit File-Deskriptoren.

**VSCode** Code Editor für Windows, Linux und macOS.

## Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

*Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI*

## Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: \_\_\_\_\_

Vorname: \_\_\_\_\_

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

### **Bildbasierte Navigation mit Neuronalen Netzen**

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original