HAW HAMBURG

Bachelorarbeit

Jan Robert Rösler

Bildbasierte Navigation mit Neuronalen Netzen

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

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 Bildaten 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

Abbildungsverzeichnis

Tabellenverzeichnis

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 [8]. 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 Austattung der Fahrzeuge (neben Ultrschallsensoren 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 Aufeinanderschichten 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 [5], 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, Machinelle 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.

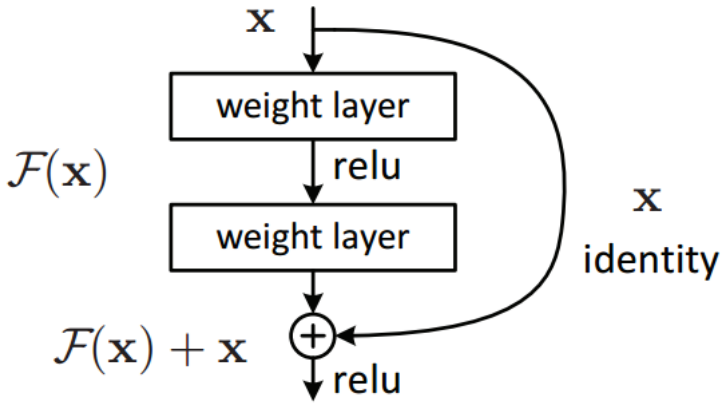


Abbildung 2.1: Residual Block
Quelle: [11]

Je tiefer ein Neuronales Netz ist, desto schwieriger wird das Trainieren aller Layer, ein Problem in diesem Zusammehang 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

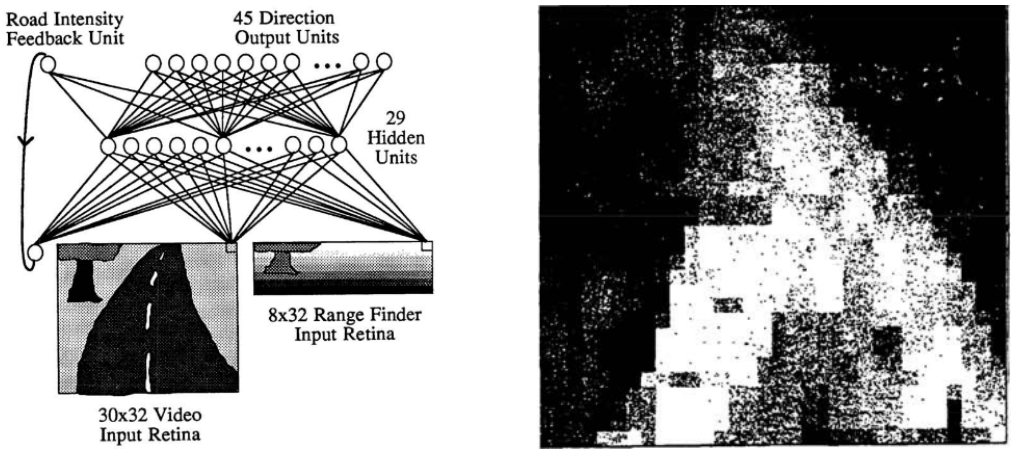
Im folgenden wird auf zwei Ansätze der Navigation mit Neuronalen Netzen eingegangen, durch Gegenüberstellung erster Versuche mit einem modernen Ansatz soll folgenden Ausarbeitungen ein Rahmen gegeben werden.

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[7]. 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 Blautstufenintensität eines Pixels bestimmt das Aktivierungslevel des Inputneurons) wird untersützt 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 Neronen 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 Entwicklung voran und...TODO
Im Jahr 2016 veröffentlicht das Technologieunternehmen NVIDIA einen eigenen Ansatz [1], basierend auf Versuchen mit dem „DARPA Autonomous Vehicle “ (DAVE) [9] wird dieser „Dave-2 “ genannt.// Daten werden hier durch Fahrten auf echten Straßen gesammelt, wofür drei Kameras in der Windschutzscheibe eines Autos angebracht und



(a) (b)

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

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.

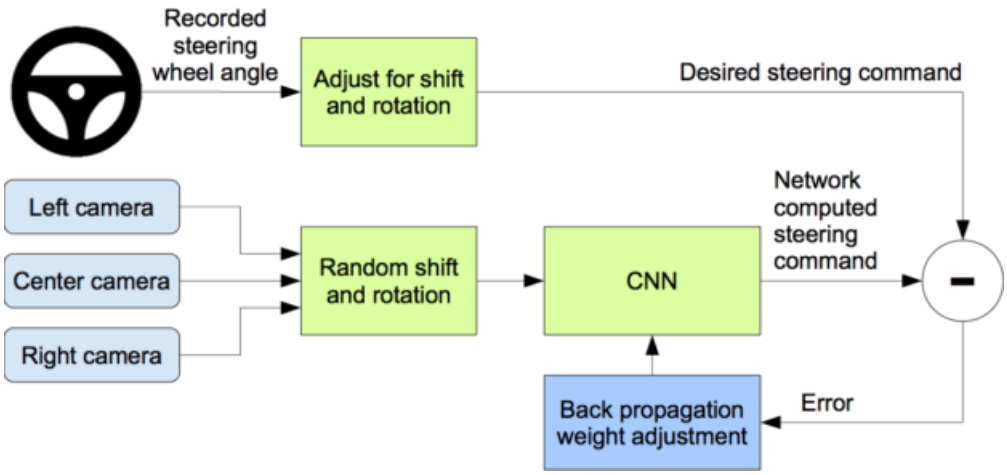


Abbildung 2.3: Komponenten des Trainings
Quelle: [10]

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 [6]. 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.

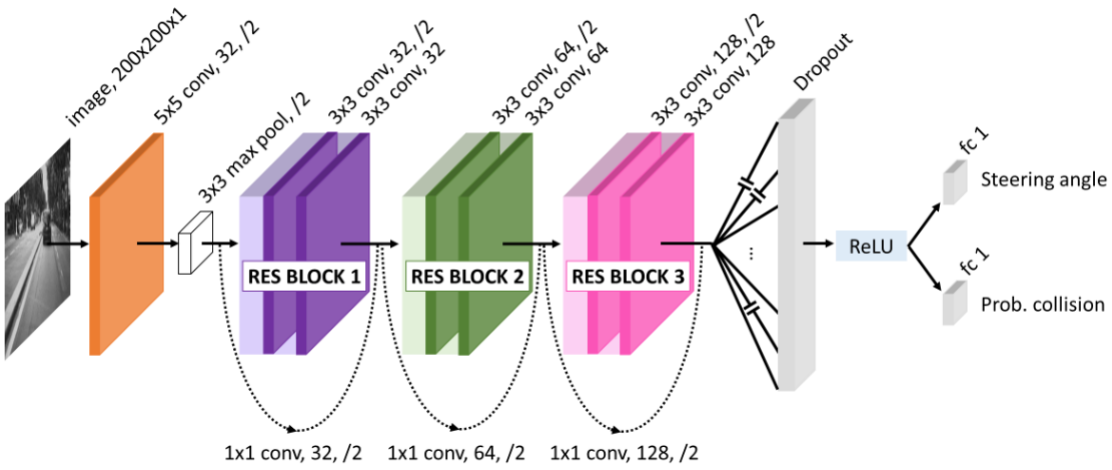


Abbildung 3.1: Architektur DroNet

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

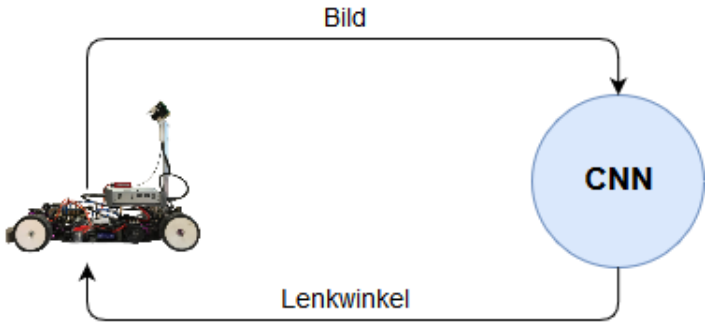


Abbildung 3.2: Einfacher schematischer Aufbau

4 Entwurf

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

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 3.2×10^5 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 UEYE 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.

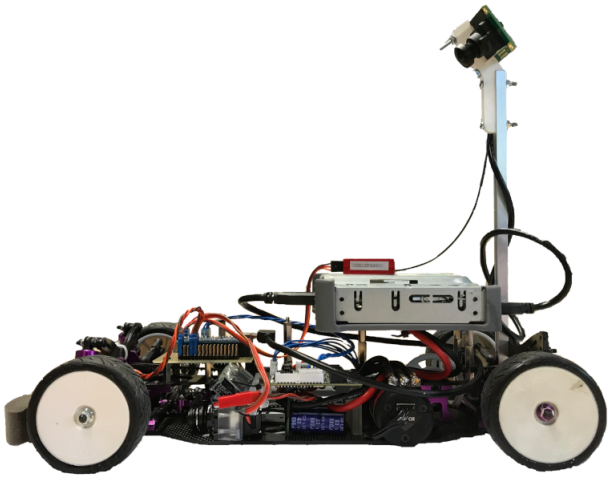


Abbildung 4.1: Das Carolo-Cup Fahrzeug

Strecke Die Teststrecke, die an der HAW zur Verfügung steht, ist ein Rundkurs mit verschiedenen Kurven und einem Kreuzungszenario. 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.

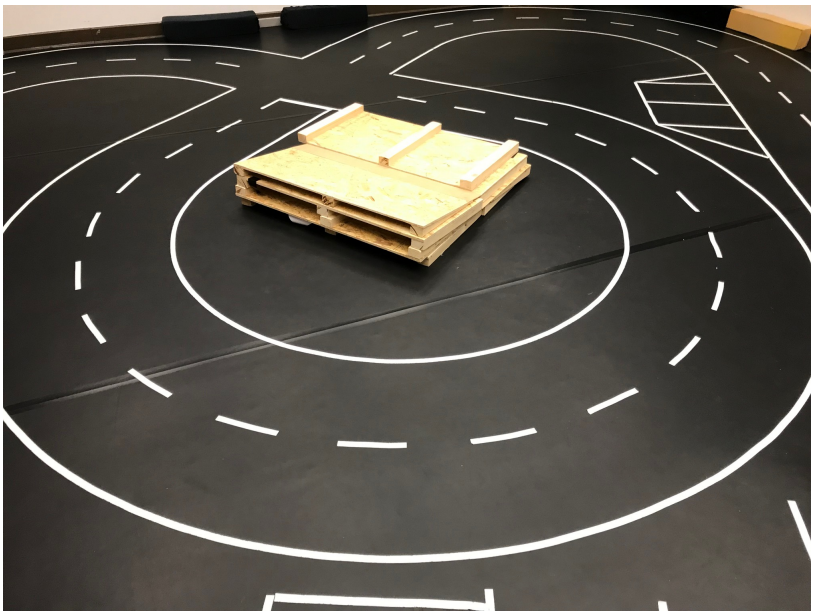


Abbildung 4.2: Ein Ausschnitt der Teststrecke

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.

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.

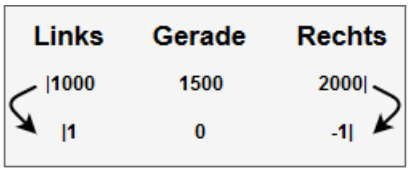
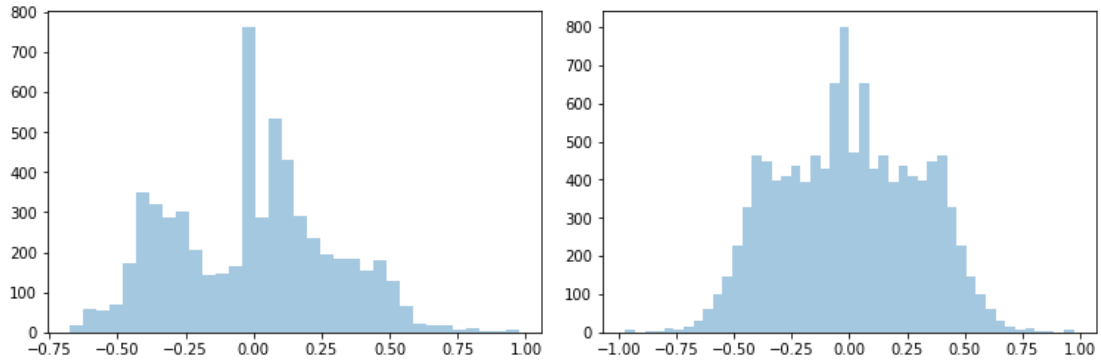


Abbildung 4.4: Skalierung der Lenkwinkel-Wertebereiche

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 Traningsbilder-Set von knapp 11.000 Bildern.



(a) (b)

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 eindeutigem Identifikator in einer Textdatei gespeichert. Die Textdatei wird für jeden Bildordner, Traning-, 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 Traningsdaten. 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 aufgezogen, 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 Traning. 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.

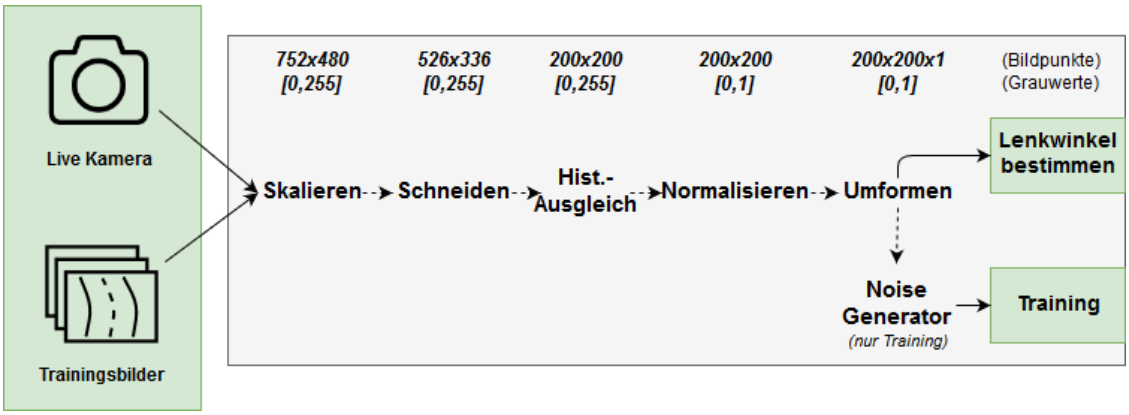


Abbildung 4.6: Bildverarbeitungsschritte

Für das Training wird auf einem Teil der Bilder ein Filter angewendet, der das Bild optisch verwischen 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.

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 4.7 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, das hier die allgemeinen Eigenschaften zu einer komplexeren Repräsentation zusammenwachsen, die gut auf die Testfahrbahn generalisiert.
Zusätzlich wird die Ausgabe des Netzes verändert. Die Kollisionswahrscheinlichkeit spielt für diese Arbeit keine Rolle und die Traningsdaten enthalten keine entsprechenden Label, daher wird jenes Ausgabe-Layer entfernt. Das Feedback für die Berechnung der Gewichtsanpassungen kommt nur vom Lenkwinkel.

Lernarchitektur Hier entsteht noch eine Erklärung der Lernarchitektur und der Netz-Parameter,die für das Fine Tuning verwendet werden

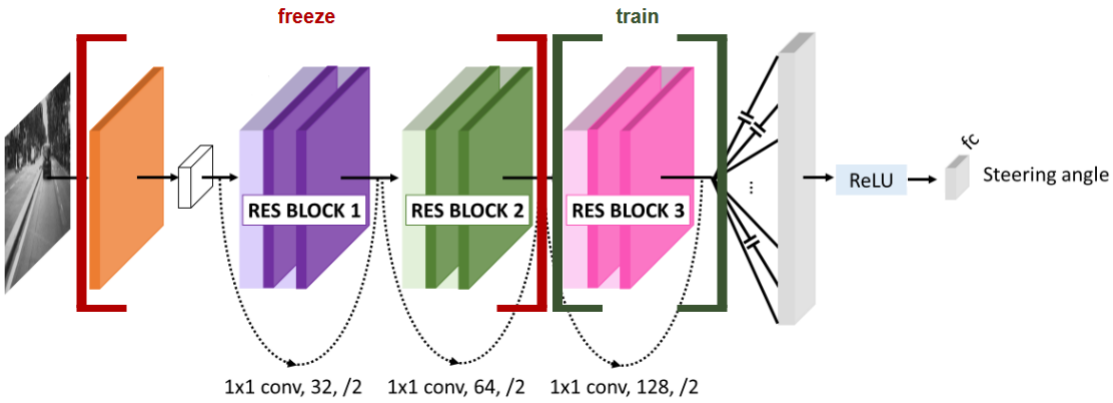


Abbildung 4.7: Angepasste Architektur

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 4.9 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 Kamerbildern. Die Steuerung soll möglichst hohen Durchsatz haben um möglichst viele Frames in einer Sekunde

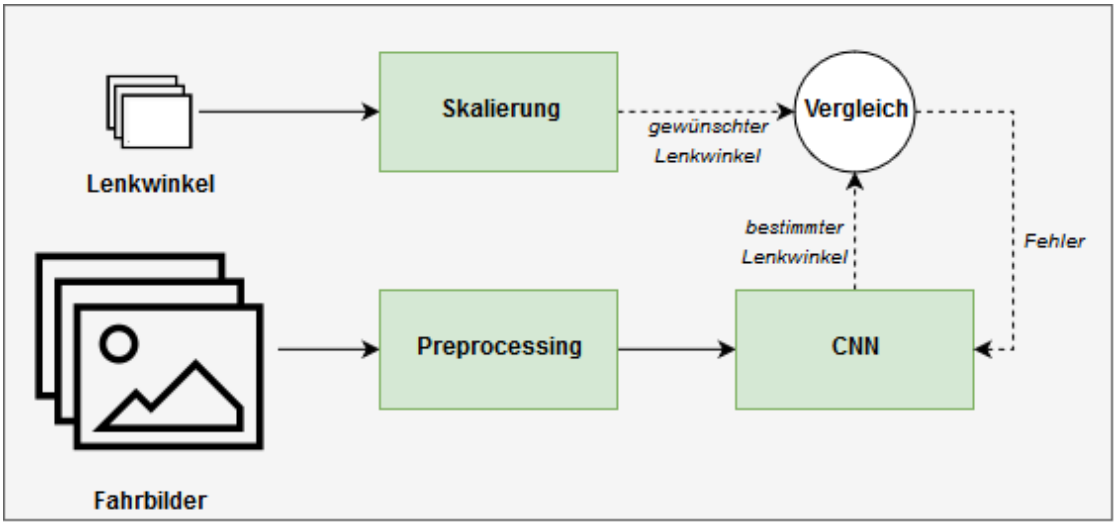


Abbildung 4.8: Struktur des Lernprozesses

zur Lenkwinkelermittung 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.

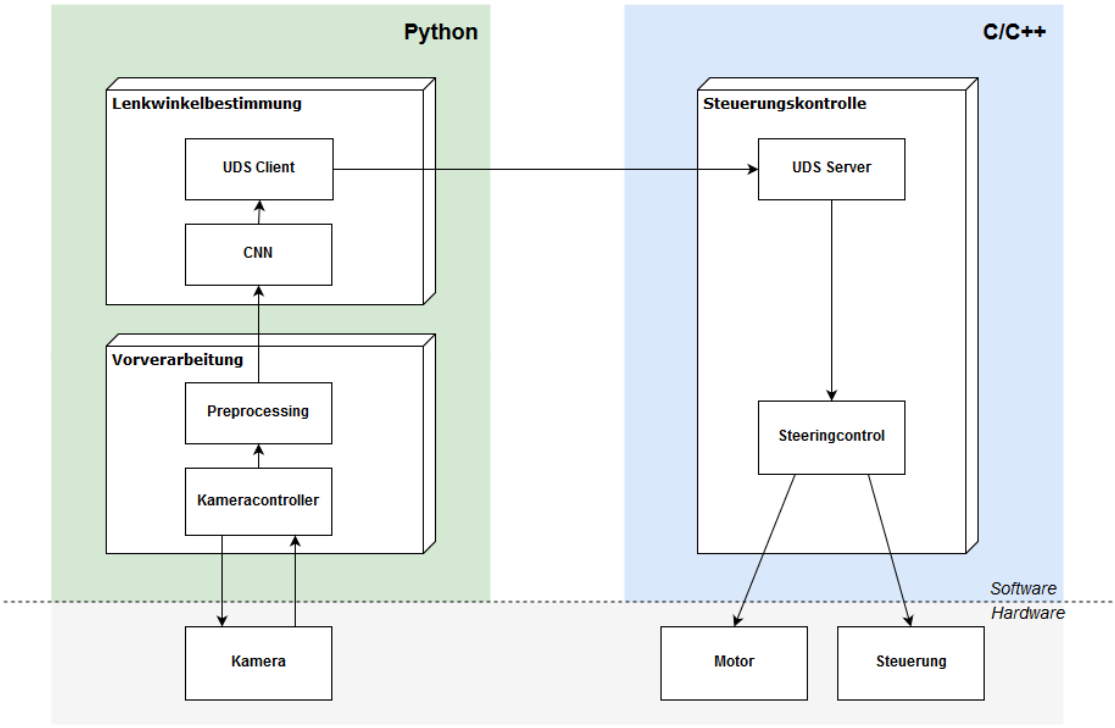


Abbildung 4.9: Komponentenstruktur und Verbindung der Neuronalen Steuerung

5 Szenarien und Auswertung

5.0.1 Metriken

Auch Geschwindigkeiten beachten, Carolo-Net performt bis 1.2 m/s

5.0.2 Testfahrten

1. Auto mit Dronet 2. Auto mit adaptiertem Netz 3. Auto mit Netz des aktuellen Carolo-Cup Teams

5.0.3 Sonderszenarien

Kreuzung

Fahrbahn wiederfinden Performance (Rechenzeit) bei prediction auf dem Fahrzeug

- Findet das Auto auf die Strecke zurück? Schräg ansetzen und autonomen Modus starten (Zeigt ob Strecke nur auswensig gelernt oder nicht, da diese blickwinkel nicht im Trainingsset enthalten sind)

5.0.4 Saliency

- Saliency Besipiele Zeigen was für Features erlernt wurden. (Eventuell auch von vorangegangenen Layern)

-Vielleicht noch ein Szenario in dem das Validation set exra schwer (Artefakte) gemacht wurde

Auf die Problematik mit Testsetgröße eingehen, kleines set daher nur letzter blockj trainiert etc....

Kurzer Blick auf Self driving car steering angle4 prediction und berkeley (large scale video sets) (vielleicht auchSPÄTER)

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, das das Fine Tuning die Lenkwinkelbestimmung entscheidend verbessert hat. Einschränkend 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 mach 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 eibbauen -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] 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

[6] Loquercio, Antonio ; Maqueda, Ana I. ; Blanco, Carlos R. D. ; Scaramuzza, Davide: Dronet: Learning to Fly by Driving. In: *IEEE Robotics and Automation Letters* (2018)

[7] Pomerleau, Dean A.: Alvinn: An autonomous land vehicle in a neural network. In: *Advances in neural information processing systems*, 1989, S. 305–313

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

Internetquellen

[I9] *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

[I10] *NVIDIA Komponenten des Steuerungssystem.* https://devblogs.nvidia.com/wp-content/uploads/2016/08/training-624x291.png. – Accessed: 2019-03-08

[I11] *Residual Block Schematischer Aufbau.* https://cdn-images-1.medium.com/max/1600/1*D0F3UitQ2l5Q0Ak-tjEdJg.png. – Accessed: 2019-03-10

Glossar

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 formalige 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