HOCHSCHULE MITTWEIDA
University of Applied Sciences

# MASTERARBEIT

Herr
**Christian Schreiber**, Dipl.-Ing. (FH)

## KI-gestützte „Follow-Me“-Funktion am Beispiel des JetRacer

Mittweida, Januar 2023

Fakultät **Ingenieurwissenschaften**

# MASTERARBEIT

## KI-gestützte „Follow-Me“-Funktion am Beispiel des JetRacer

Autor:
**Christian Schreiber**

Studiengang:
Elektrotechnik - Automation

Seminargruppe:
EA20wV-M

Erstprüfer:
Prof. Dr.-Ing. Daniel Kriesten

Zweitprüfer:
Prof. Dr.-Ing. Jan Thomanek

Einreichung:
Mittweida, 17.01.2023

**Bibliografische Beschreibung:**

Schreiber, Christian:
KI-gestützte „Follow-Me“-Funktion am Beispiel des JetRacer. – 2023. – 59 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Ingenieurwissenschaften, Masterarbeit, 2023.

**Referat:**

Diese Abschlussarbeit untersucht die Möglichkeit, KI-gestützt eine „Follow-Me“-Funktion am Beispiel eines JetRacers darzustellen.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Abkürzungsverzeichnis

**ABS** . . . . . Antiblockiersystem

**ACC** . . . . . Adaptive Abstands- und Geschwindigkeitsregelung (engl. Adaptive cruise control)

**ADC** . . . . . Analog-Digital-Converter

**AFS** . . . . . Adaptive Lenkung

**AI** . . . . . Artificial Intelligence

**ANN** . . . . . Artificial Neural Networks

**CNN** . . . . . faltende neuronale Netze (engl. Convolutional Neural Networks)

**CoCa** . . . . . Contrastive Captioners - Objekterkennung Algorithmus

**CPU** . . . . . Zentraleinheit (engl. Central Processing Unit)

**CUDA** . . . . . Compute Unified Device Architecture

**DAG** . . . . . gerichteten azyklischen Graphen (engl. Directed Acyclic Graph)

**DNN** . . . . . tiefen neuronalen Netzen (engl. Deep Neural Network)

**EPS** . . . . . Electric Power Steering

**ESP** . . . . . Elektronisches Stabilitätsprogramm

**FOV** . . . . . Sichtfeld (engl. Field of View)

**FP16** . . . . . 16-Bit-Fließkommazahl (engl. floating-point with 16 bit)

**fps** . . . . . Bilder pro Sekunde (engl. Frames per Second)

**GPU** . . . . . Grafikeinheiten (engl. Graphics Processing Unit)

**GUI** . . . . . grafische Benutzeroberfläche (engl. Graphical User Interface)

**HD** . . . . . High-Definition

**HOG** . . . . . Histogram of Oriented Gradients

**IDE** . . . . . Integrated Development Environment

**INT8** . . . . . 8-Bit-Ganzzahl (engl. Integer with 8 bit)

**KI** . . . . . künstliche Intelligenz

**KNN** . . . . . künstliche neuronale Netze

**ML** . . . . . Maschinelles Lernen (engl. Machine Learning)

**MSE** . . . . . mittleren quadratischen Fehlers (engl. Mean Squared Error)

**PC** . . . . . Personal Computer

**Pkw** . . . . . Personenkraftwagen

**ReLU** . . . . . Rectified Linear Unit

**SAE** . . . . . Society of Automotive Engineers

**SDK** . . . . . Software Development Kit

**SGD** . . . . . Stochastischen Gradientenabstieg (engl. Stochastic Gradient Descent)

**SSH** . . . . . Secure Shell

**SVM** . . . . . Support Vector Machine

**UNECE** . . . . . Wirtschaftskommission für Europa der Vereinten Nationen (engl. United Nations Economic Commission for Europe)

**VDA** . . . . . Verband der Automobilindustrie

# 1 Einleitung

Autonomes Fahren gehört womöglich zu den prägendsten sozio-ökonomischen Entwicklungen der kommenden Jahrzehnte. Dabei haben selbstfahrende Fahrzeuge eine enorme gesellschaftliche Tragweite. Denn diese haben das Potenzial, zu verändern, wie Menschen arbeiten und leben werden. Autonome Autos werden für alle gesellschaftliche Schichten relevant sein. Dabei zeigen die Entwicklungen der vergangenen Jahrzehnte, dass sich dieser Entwicklung kaum direkt oder indirekt entzogen werden kann. [1]

Der motorisierte Individualverkehr, vordergründig Personenkraftwagen (Pkw) und Krafträder, nahm in Deutschland zwischen 1991 und 2019 um etwa 28,5 % zu. [2]

Damit geht eine steigende Belastung der Verkehrsinfrastruktur einher, die teilweise zu einer Überlastung der Infrastruktur führt. Als Resultat kommt es zu steigenden Emissionen, erhöhtem Stauaufkommen, mangelnde Parkmöglichkeiten sowie weiteren negativen Beeinträchtigungen bis zu einer höheren psychischen Belastung der Einwohner. Gleichzeitig gilt jedoch, dass ein Pkw ca. 95 % der Zeit steht und dementsprechend nur in 5 % der Zeit „effektiv“ genutzt wird. [1]

Als einer der wichtigsten Punkte ist noch die Sicherheit des motorisierten Individualverkehrs zu sehen. Die Zahlen der Verkehrstoten sinken zwar seit den 1970er-Jahren immer weiter. [3] Der Pkw gilt im Vergleich zu anderen Verkehrsmitteln, wie Straßenbahn, Bus, Eisenbahn, Schiff oder Flugzeug, trotzdem als das gefährlichste Verkehrsmittel, abgesehen von den motorisierten Zweirädern. [4]

Zusammengefasst kann festgehalten werden, dass mittels autonomer Fahrzeuge die Infrastruktur entlastet, Emissionen reduziert, die Sicherheit der Bevölkerung gesteigert und die Lebensqualität der Menschen gehoben werden kann. Daraus resultierend kann auch festgehalten werden, dass sich nicht die Frage stellt, ob es autonome Fahrzeuge geben wird, sondern vielmehr, wann sie sich durchsetzen können und dadurch das selbst gelenkte Fahrzeug verdrängen. Die eingesetzten Techniken, die motorisierte Fahrzeuge autonom fahren lassen, sollen im nächsten Kapitel betrachtet werden.

Wie bereits verdeutlicht, wird das autonome Fahren sich zunehmend etablieren. Aufgrund der Tatsache, dass es eine Vielzahl von Fahraufgaben gibt, z. B. Ein-, Ausparken, Beschleunigen, Abbremsen, Abstandhalten, Verarbeitung von Verkehrszeichen, Abbiegevorgänge, Routenplanung, und viele weitere, die ein autonomes Fahrzeug bewältigen muss, lässt sich diese Komplexität durch Aufteilen in Teilaufgaben reduzieren. So soll im Rahmen dieser Masterarbeit die „Follow-Me“-Funktion anhand eines JetRacer umgesetzt werden. „Follow-Me“ bedeutet in Kontext dieser Masterarbeit, dass ein Fahrzeug einem anderen Fahrzeug folgt, ohne dem vorausfahrenden Fahrzeug aufzufahren, auch wenn es bis zum Stillstand abbremst oder auch zu beschleunigen, wenn das vorfahrende Fahrzeug beschleunigt. Gleichzeitig darf das folgende Fahrzeug, auch EGO-Fahrzeug genannt, nicht die Spur verlassen und muss der Fahrbahn folgen. Die Umsetzung dieser Fahraufgabe kann nach konventionellem Ansatz umgesetzt werden, der u. a. im Abschnitt 2.2 erläutert wird. Alternativ dazu kann ein „KI-gestützter“ Ansatz betrachtet werden, dabei steht die Abkürzung KI für künstliche Intelligenz. Dieser Ansatz wird in Abschnitt 2.3 dargelegt. Ferner wird in dem Abschnitt 2.4 das Gebiet der KI eingeordnet und spezielle auf künstliche neuronale Netze (KNN) eingegangen, da diese im weiteren Verlauf für die Implementierung der „Follow-Me“-Funktion mit einem KI-Ansatz benötigt werden.

Für die Umsetzung der Fahraufgaben wird im Rahmen dieser Masterarbeit kein Pkw als solches genutzt, sondern ein JetRacer, wobei es sich um eine Art ferngesteuertes Auto handelt. Die Beschreibung der Hardware dieses kleinen Fahrzeugs erfolgt im Kapitel 3.

Im Rahmen dieser Arbeit soll der KI-gestützte Ansatz zur Umsetzung der „Follow-Me“ Fahraufgabe genutzt werden. Die zu implementierende „Follow-Me“-Funktion kann unabhängig von dem Ansatz in zwei weitere Teilaufgaben untergliedert werden. Wie bereits im vorherigen Absatz geschildert, muss das Fahrzeug zum einen der Spur folgen (1. Teilfahraufgabe) und zum anderen den Abstand zum vorausfahrenden Fahrzeug halten (2. Teilfahraufgabe). Diese Untergliederung sowie die eigentliche Umsetzung, wird in dem Kapitel 4 erläutert.

Zum Abschluss der Arbeit wird in Kapitel 5 noch einmal auf die Ergebnisse eingegangen und ein Ausblick gegeben.

# 2 Stand der Technik

Um zu verstehen, was unter dem Begriff „autonomes Fahren“ verstanden wird, stellt dieses Kapitel die Grundlagen zum autonomen Fahren und der dafür erforderlichen Systeme her. Dabei soll auch auf die Standardisierung der von der Society of Automotive Engineers (SAE) eingeführte Einteilung des autonomen Fahrens eingegangen werden.

Des Weiteren wird erläutert, welche groben Ausprägungs- und Umsetzungsarten aktuell in den Fahrzeugen genutzt werden, aber auch in der Entwicklung stehen. Hierbei soll das autonome Fahren im klassischen Sinne vorgestellt werden. Im Vergleich dazu wird der KI-gestützte Ansatz bzw. die Systeme, die KNN nutzen, vorgestellt und was diesen von dem klassischen Ansatz unterscheidet.

Den Abschluss dieses Kapitels stellt die Einführung in die Grundlagen der KI und der neuronalen Netze dar.

## 2.1 Autonomes Fahren nach SAE

Um zu klären, was „Autonomes Fahren“ ist, wird im Vorfeld das „einfache/selbstständige“ Fahren analysiert. Das aktive Teilnehmen am Straßenverkehr als Fahrer stellt eine komplexe Überwachungs- und Regelungsaufgabe dar, für deren Gelingen bei dem aktuellen Stand der Technik der Fahrer voll verantwortlich ist.

Es besteht die Möglichkeit, die Fahraufgabe nach einem funktionalen Ansatz in folgende Ebenen zu unterteilen:

- Navigation
- Bahnführung
- Stabilisierung

Anhand des Ausgangsziels der Fahraufgabe einen vom Startpunkt verschiedenen Zielort zu erreichen, erschließt sich mit der Planung einer Route die Ebene der Navigation. Die Route ist notwendig, um an Verzweigungspunkten des Straßennetzes die zielführende Abzweigung zu wählen. Die Wahl der Fahrroute hängt insbesondere vom Straßennetz und der zur Verfügung stehenden Zeit für die Fahrt ab. Eine zulässige Geschwindigkeit oder die aktuelle Verkehrslage, aufgrund von Baustellen oder Staus, können weitere Einflussfaktoren auf die Fahrroute darstellen. Die benötige Zeit für die Planungsarbeit auf der Navigationsebene bedarf, meist die gesamte Fahrt. Die Planung auf dieser Ebene nimmt damit, verglichen mit den weiteren Ebenen, den größten Zeitaufwand in Anspruch. [5]

Die Führungs- bzw. Bahnführungsebene ist die Ebene, auf welcher eine Steuerung des Fahrzeugs in Form eines offenen Regelkreises stattfindet. Diese umfasst die Zielgeschwindigkeit sowie die Bestimmung eines Soll-Kurses, also z. B. die Auswahl der Soll-Spur. Zusätzlich zu den Einflüssen aus der Navigationsebene wird die Fahrzeugführung auf Bahnführungsebene durch die aktuelle Verkehrssituation beeinflusst. Auf dieser Ebene haben die Anzahl vorhandener Spuren, aber auch weitere Verkehrsteilnehmer oder Ampelphasen großen Einfluss. Der entsprechend beeinflusste Zeitbereich auf der Bahnführungsebene liegt im Bereich von mehreren Sekunden. [6]

Die Ebene der Stabilisierung beschreibt die Ausführung, der in den beiden anderen Ebenen geplanten Vorgaben. Auf dieser Ebene handeln fahrzeugführende Personen im Bereich von Millisekunden und folgen den Vorgaben der Zielgeschwindigkeit und des Soll-Kurses. [6]

Die Fahraufgabe auf Basis der funktionalen Unterteilung in Unteraufgaben ist anhand der Abbildung 2.1 dargestellt.

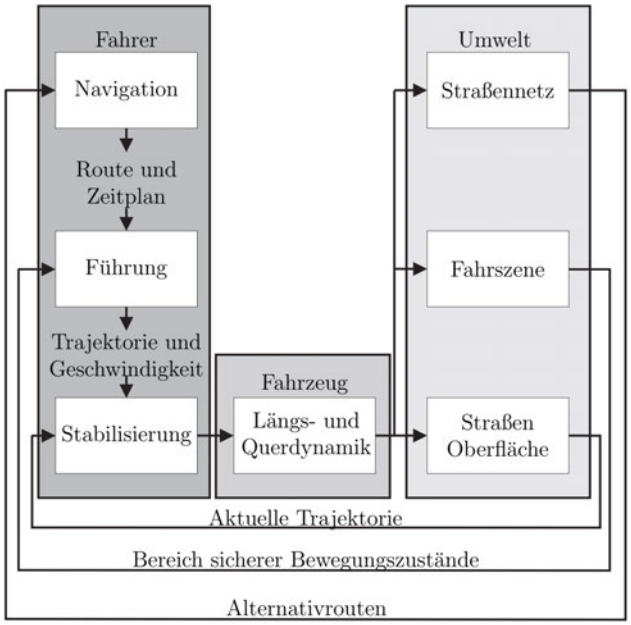


**Abbildung 2.1:** Unterteilung der Fahraufgabe nach [6]

Nachdem die Frage geklärt wurde, was das „einfache bzw. selbstständige“ Fahren bedeutet und welche Komplexität bereits in dieser formal einfachen Tätigkeit liegt, kann sich dem autonomen Fahren gewidmet werden.

Unter autonomen Fahren wird das automatisierte Fahren, also der autonome Ablauf der zuvor dargestellten komplexen Fahraufgabe verstanden. Das bedeute, dass die Unteraufgaben Navigation (Routenplanung), Führung (z. B. Spurwechsel) und Stabilisierung (Fahrzeug innerhalb der physikalischen Vorgaben halten) ohne menschliches Eingreifen realisiert werden. Konkret bedeutet dies, dass das Fahrzeug eine Route wählt und z. B. an Kreuzungen selbstständig unter Einhaltung der rechtlichen und physikalischen Vorgaben abbiegt.

Die SAE unterscheidet dabei zwischen fünf Stufen des automatisierten Fahrens. Wobei nur die Stufen 4 und 5 das reine autonome Fahren abbilden. In der nachfolgenden Abbildung 2.2 sind die Stufen mit der SAE-Bezeichnung und der deutschen Bezeichnung nach dem Verband der Automobilindustrie (VDA) dargestellt.

Wie aus der Abbildung 2.2 hervorgeht, findet beim assistierten Fahren (Level 1) und beim teilautomatisierten Fahren (Level 2) lediglich eine Unterstützung in der Längs- und/oder Querführung des Fahrzeugs statt, also in dem Unteraufgabenbereich „Stabilisierung“. Die gesamte Fahraufgabe und die daraus resultierende Verantwortung liegt hierbei bei der fahrzeugführenden Person. Diese Systeme sind bereits Stand der Technik, z. B. Abstandsregeltempomat oder Spurhalteassistent.

Sobald der Fahrer sich kurzzeitig von der Fahraufgabe abwenden kann, handelt es sich um ein hochautomatisiertes Fahrerassistenzsystem (Level 3), das die Unteraufgaben „Stabilisierung“ und „Führung“ übernimmt. Dabei kann die fahrzeugführende Person die Verantwortung zeitweise an das

| Stufe | Stufe 0 | Stufe 1 | Stufe 2 | Stufe 3 | Stufe 4 | Stufe 5 |
|---|---|---|---|---|---|---|
| **Bezeichnung (BASt/VDA)** (SAE-Bezeichnung) | **Nicht automatisiert** (no automation) | **Assistiert** (driver assistance) | **Teil-automatisiert** (partial driving automation) | **Hoch-automatisiert** (conditional driving automation) | **Voll-automatisiert** (high driving automation) | **Fahrerlos/autonom** (≠ full driving automation) |
| **Ausführung der Fahraufgabe** | **Fahrer** | **Fahrer,** unterstützt durch System hinsichtlich Längs- **oder** Querführung | **System** führt Längs- **und** Querführung aus | **System** führt Längs- **und** Querführung aus | **System** führt Längs- **und** Querführung aus | **System** führt Längs- **und** Querführung aus |
| **Überwachung der Fahraufgabe** | Dauerhaft durch **Fahrer** | Dauerhaft durch **Fahrer** | Dauerhaft durch **Fahrer** | Während der automatisierten Fahrt durch **System** | Während der automatisierten Fahrt durch **System** | Dauerhaft durch **System** |
| **Rückfallebene im Fehlerfall** | Keine | **Fahrer** | **Fahrer** | **Fahrer** wird mehrmals zur Übernahme aufgefordert | **System** | **System** |
| **Anforderungen an den Fahrer** | Dauerhafte **Ausführung** der Fahraufgabe | **Dauerhafte Überwachung** und **Ausführung** der Quer- **oder** Längsführung | **Dauerhafte Überwachung** und jederzeit **Bereitschaft** zur vollständigen Übernahme | **Bereitschaft,** die Fahraufgabe nach mehreren Aufforderungen zu übernehmen | **Keine,** Fahrer kann auf Wunsch das System abschalten und selbst fahren | **Kein Fahrer im Fahrzeug,** ggf. Lotse außerhalb des Fahrzeugs |
| **Beispiel** | Fahrer fährt | Abstandsregeltempomat | Stauassistent unterstützt Längs- und Querführung auf Autobahnen | Autobahn-System übernimmt zeitweise Längs- und Querführung auf Autobahnen | Autobahn-System übernimmt Längs- und Querführung auf Autobahnen | Fahrerloses Shuttle auf definierten Strecken in Stadt XY |

**Abbildung 2.2:** Stufen der Automatisierung [7]

Fahrzeug abgeben. Die fahrzeugführende Person kann in diesem Fall andere Aufgaben während der Fahrt übernehmen, muss aber wach bleiben und nach Aufforderung die Fahraufgabe wieder wahrnehmen können. Dieses System findet aktuell in den sehr hochpreisigen Fahrzeugsegmenten Einzug. Als erste Fahrzeuge, die offiziell hochautomatisiert bis 60 km/h auf deutschen Straßen fahren dürfen, sind die Mercedes S-Klasse und der Mercedes EQS zu nennen. Der dafür notwendige rechtliche Rahmen wurde erst im Jahr 2021 verabschiedet und lässt das Fahren bis 60 km/h zu. [8] Um zügiger auf deutschen Autobahnen hochautomatisiert fahren zu dürfen, bedarf es einer weiteren rechtlichen Anpassung, die im Rahmen der Gesetzgebung der Wirtschaftskommission für Europa der Vereinten Nationen (engl. United Nations Economic Commission for Europe) (UNECE) für das Jahr 2023 geplant ist. Dadurch soll die aktuell zulässige Geschwindigkeit für Level-3 Systeme von 60 km/h auf 130 km/h angehoben werden. [9]

Die Steigerung des hochautomatisierten Fahrerassistenzsystems stellen voll automatisierte Fahrerassistenzsysteme (Level 4) dar, bei denen keine fahrzeugführende Person mehr benötigt wird. Dementsprechend wird die eingangs geschilderte Fahraufgabe inkl. aller Unteraufgaben „Navigation, Führung und Stabilisierung“ übernommen. Auch im Grenz- oder Fehlerfall des Systems muss das System die Situation komplett in eigener Verantwortung lösen. Für potenziell fahrzeugführende Person besteht weiterhin die Möglichkeit, das System abzuschalten und selbstständig zu fahren. Für diese Stufe des autonomen Fahrens sind neben technischen Entwicklungen auch noch weitere Rechtsprechungen erforderlich. Aktuell gibt es noch kein System, das die Anforderung an ein voll automatisiertes Fahren erfüllt.

Das autonome Fahren (Stufe 5) stellt die maximale Ausbaustufe in Bezug auf die Automatisierung dar. Ein autonomes Fahrzeug kann sämtliche Fahraufgaben komplett selbstständig, ohne das Zutun einer fahrzeugführenden Person, erbringen. In autonomen Fahrzeugen kann mitunter die Instrumentierung nicht mehr vorhanden sein, um die Fahraufgabe durch eine fahrzeugführende Person umsetzten zu lassen. Diese Fahrzeuge könnten somit komplett fahrerlos betrieben werden.

Im Rahmen dieser Arbeit soll das Fahrerassistenzsystem „Follow-Me“ mit einem KI-gestützten Ansatz umgesetzt werden. Dafür ist es erforderlich, zu verstehen, welche Fahrerassistenzsystem es gibt und wie konkret die „Follow-Me“-Funktionalität aussieht. Da im weiteren Verlauf auf den KI-gestützten Ansatz eingegangen wird, ist es dienlich, auf den nicht KI-gestützten Ansatz, den „klassischen Ansatz“ einzugehen.

In dem folgenden Abschnitt wird die Frage geklärt, wie Fahrerassistenzsysteme aufgebaut sind, was diese benötigen, um ihre Assistenzfunktion umzusetzen und welche Fahrerassistenzsysteme bereits vorhanden sind.

## 2.2 Fahrerassistenzsysteme

Fahrerassistenzsysteme existieren in verschiedensten Ausprägungen und Funktionstiefen. Es gibt diese bereits seit Ende der 1970er-Jahre. Das erste Assistenzsystem ist das Antiblockiersystem (ABS), das im Falle einer Vollbremsung das Blockieren der Räder verhindert. Dadurch wird die Lenkbarkeit des Fahrzeugs beibehalten, was zu einer Steigerung der Fahrzeugkontrollierbarkeit und Sicherheit beiträgt.

Aufgrund der Vielzahl von Assistenzsystemen (ABS, Elektronisches Stabilitätsprogramm (ESP), Adaptive Lenkung (AFS), Spurverlassenswarnung, Spurhalteassistent, Spurwechselassistent, Adaptive Abstands- und Geschwindigkeitsregelung (engl. Adaptive cruise control) (ACC) u.w.) wird im Rahmen dieser Arbeit nur auf die Assistenten eingegangen, die als Basis für die Implementierung der „Follow-Me“-Funktion benötigt werden.

Ein Fahrerassistenzsystem benötigt unterschiedliche Sensoren (siehe Unterabschnitt 2.2.1), eine elektronische Steuerung, die die Sensordaten auswertet (siehe Unterabschnitt 2.2.2) und anschließend Steuersignale sendet – entweder an Lautsprecher und Anzeigen, um den Fahrer zu warnen oder an Aktuatoren, um aktiv in die Fahrzeugsteuerung einzugreifen (siehe Unterabschnitt 2.2.3).

So benötigt u. a. das ABS einzelne Drehzahlsensoren (Sensor) an den Rädern, um z. B. einen Abfall der Raddrehzahl gegenüber den anderen Rädern festzustellen und daraus schlussfolgernd den Bremsdruck am jeweiligen Rad kurzzeitig zu reduzieren, was das ABS-Modul übernimmt (Aktuator).

Grundsätzlich lässt sich ein Fahrerassistenzsystem in drei zentrale Aufgaben unterteilen. Diese werden in den nachfolgenden Abschnitten beschrieben. [10]

### 2.2.1 Wahrnehmung – Messen und Erkennen

Unter den Begriff der Wahrnehmung fällt im Kontext der Assistenzsysteme die Aufnahme von Sensordaten (Messen) sowie deren Verarbeitung und Interpretation (Erkennen). Im folgenden Abschnitt wird eine Auswahl von Sensoren vorgestellt, wie diese für die erwähnten Fahrerassistenzsysteme grundsätzlich notwendig sind.

#### 2.2.1.1 Messen – Sensorik in Fahrzeugen

Sensoren erfassen Sollwerte (z.B. Lenkradwinkel), Betriebszustände (z. B. Raddrehzahl) oder Lichtintensitäten (z. B. Bildsensor einer Kamera). Dabei wandeln sie physikalische Größen in elektrische Signale um. Es gibt eine Vielzahl von Sensoren, die u. a. für Fahrerassistenzsysteme genutzt werden. Im Rahmen dieser Arbeit wird der Fokus auf diejenige Sensorik gelegt, die zur Implementierung der „Follow-Me“-Funktion notwendig ist. Für diese Funktion wird auf die Fahrerassistenzfunktionen des Spurhalteassistenten und Abstandsregeltempomat zurückgegriffen. Beide bauen auf Subassistenzsystemen wie ABS oder ESP aber auch dem Spurverlassenwarner auf.

Die aufgezählten Systeme nutzen u. a. die nachfolgenden Sensoren als Eingangsgrößen und werden daher im Folgenden vorgestellt.

**Basis Fahrdynamik Sensoren** Hierunter fallen alle Sensoren zur Bestimmung der eigenen Fahrdynamik, von der reinen Geschwindigkeitsbestimmung über den Lenkwinkel bis zu Beschleunigungs- und Gierratensensoren. Diese Systeme werden nicht näher erläutert und können in [6] nachgelesen werden.

**Radar (engl. Radio Detection and Ranging)** Wird in Pkws vordergründig für die Abstandsmessung genutzt. Die vom Radar ausgesendeten elektromagnetischen Wellen werden an Oberflächen aus Metall oder anderen reflektierenden Materialien, die sich im Sensorsichtbereich befinden, reflektiert und können vom Empfangsteil des Radars wieder aufgenommen werden. Aus der Laufzeit dieser elektromagnetischen Wellen kann der Abstand zu den Objekten gemessen werden. Zur Messung der Relativgeschwindigkeit bietet sich der Dopplereffekt an. Die Radartechnik stellt somit eine Möglichkeit dar, das Umfeld eines Kraftfahrzeugs zu erfassen und Abstände von Objekten im Sichtbereich des Sensors zu bestimmen. Für die Anwendungen im Straßenverkehr stehen aktuell vier Frequenzbänder zur Verfügung, wobei der 76,5 GHz-Bereich dominierend ist. Dabei nutzt der Radar im Wesentlichen Frequenzen zwischen 76 - 77 GHz. Neben der Abstandsmessung kann mittels eines Radars auch die Lage (Winkel) eines Radarobjekts bestimmt werden. Um dies zu erreichen, muss die Radarwelle in verschiedene Richtungen abgestrahlt werden, weitere Informationen werden in [11] dargelegt.

**Lidar (engl. Light Detection and Ranging)** Ist in Pkws vorwiegend für die laterale Objekterkennung im nahen bis mittleren Entfernungsbereich im Einsatz. Dabei arbeiten Lidar-Sensoren im Prinzip wie Radarsensoren, mit dem Unterschied, dass diese elektromagnetischen Wellen im infraroten Wellenlängenbereich zwischen 800 und 1000 nm benutzen, anstatt Mikrowellen im mm-Bereich. Lidar-Strahlen werden durch Nebel und schlechte Sichtverhältnisse oder Gischt mitunter erheblich gedämpft. Die Messreichweite kann dadurch also entsprechend reduziert werden, was dieses Sensorsystem für Sicherheitsanwendungen weniger geeignet erscheinen lässt als Radarsensoren. Der Lidar-Sensor emittiert modulierte Infrarotstrahlung, die von einem Objekt reflektiert und von einer oder mehreren Fotodioden im Sensor empfangen wird. Mittels einer Phasendifferenz oder Laufzeitdifferenz kann die Entfernung zum Objekt berechnet werden. Die Modulationsart bestimmt dabei stark das Signal zu Rauschverhältnis, wobei die Pulsweitenmodulation die besten Ergebnisse erreicht. Der Lidar misst die Objektgeschwindigkeit nicht direkt, sondern durch Differenzierung des Entfernungssignals, wodurch eine Verzögerung durch die Verarbeitung entsteht. Dem gegenüber steht jedoch die gute laterale Auflösung, die dem Radarsensor deutlich überlegen ist. Weiterführende Informationen zu dem Sensorsystem können in [6] nachgelesen werden.

**Videotechnik/Kamera** Den höchsten Informationsgehalt für Menschen besitzen Bilder, dementsprechend ist es naheliegend, dass für die Entwicklung von Fahrerassistenzsystemen auf Kamerasysteme zur Bilderfassung zurückgegriffen wird. Die Systeme dienen u. a. der Objekterkennung, z. B. für den Verkehrszeichenassistenten oder den Spurhalteassistenten, der die Fahrbahnmarkierung detektiert. Eine Bildquelle, z. B. ein beleuchteter Gegenstand, wird mittels einer Optik auf einem Bildsensor abgebildet. Der Bildsensor wandelt die einfallende Strahlung in seinen einzelnen Bildpunkten (Imager) in eine elektrische Ladung um, die elektronisch verarbeitet werden kann. Bei einer Analog-Kamera wird die Information mit einem Analog-Digital-Converter (ADC) in digitale Signale zur weiteren Bildverarbeitung umgewandelt. Anschließend kann, z. B. im Rahmen der Bildverarbeitung, eine Verkehrszeichenerkennung erfolgen, die als Ausgabe eine Information an den Fahrer übermittelt, welches Verkehrszeichen aktuell gilt. Es können dabei jedoch auch andere Aktuatoren als ein Bildschirm zum Einsatz kommen, wie ein aktiver Lenkeingriff, um die Spur halten zu können. Weiterhin sei erwähnt, dass es auch neben einer Monokamera ebenfalls Stereokamerasysteme gibt.

Der Vorteil dieser Kamerasysteme ist, dass mittels der beiden genau aufeinander abgestimmten Kameralinsen, sich Tiefeninformationen aus dem Bild bestimmen lassen, die mittels einer Monokamera nicht direkt möglich sind. Weitere Informationen zum Aufbau der Optik, den verfügbaren Bildsensoren und Kamera-Architekturen kann in [6] nachgelesen werden.

Eine Übersicht über die im Fahrzeug verwendeten System zur Umfelderfassung ist anhand der Abbildung 2.3 dargestellt. Entsprechend den Anforderungen und der Aufgabe, kommen, wie bereits gezeigt, verschiedene Sensoren zum Einsatz. In der Abbildung 2.3 wird dabei nach der Reichweite für eine Abstandsmessung unterschieden:

1. Fernbereichsradar (77 GHz) mit Reichweite < 200 m
2. Fern-/Nahbereichs-Infrarotsichtsystem mit Reichweite < 150 m (Nachtsichtbereich)
3. Außenbereich-Video (mittlerer Bereich < 80 m)
4. Nahbereichsradar 24 GHz (Nahbereich < 20 m)
5. Innenraum-Video
6. Ultraschall-Sensorik (Ultranahbereich < 2,5 m)

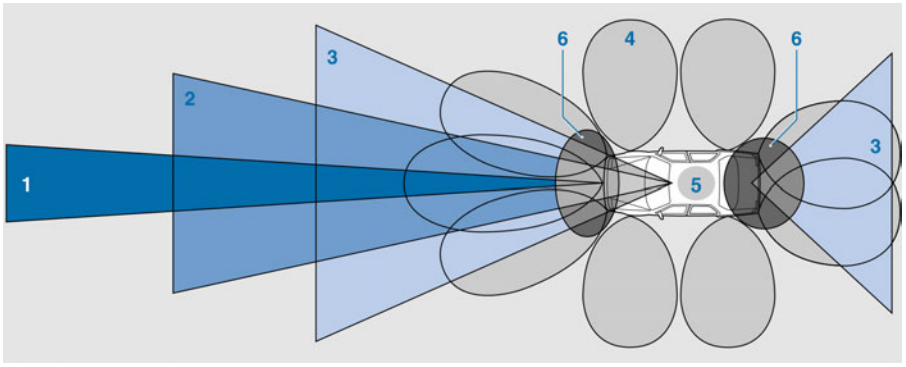


**Abbildung 2.3:** Übersicht Nah- und Fernbereichs-Abstandmessung im Pkw [11]

Die Aufbereitung der Sensordaten als solches wird in diesem Kapitel außer Acht gelassen, da sich u. a. die Methoden entsprechend der verschiedenen Sensorsysteme unterscheiden. Hierbei kann auf das Handbuch [6] verwiesen werden. Nachdem in diesem Absatz die Umfelderfassung mittels Sensorik thematisiert wurde, soll es in dem folgenden Abschnitt um die Auswertung der Sensordaten, eine Lokalisierung, Objekterkennung und Objekttracking anhand der Sensordaten gehen.

#### 2.2.1.2 Erkennen – Verarbeitung und Interpretation

In diesem Abschnitt wird die Verarbeitung bis hin zur Situationsanalyse anhand der Sensordaten geschildert, um letztlich Handlungen (siehe Unterabschnitt 2.2.3) in Form von Assistenzfunktionen umzusetzen. Dabei kann die Sensordatenverarbeitung in verschiedene Teile unterteilt werden, welche im Folgenden aufgeführt sind. [10]

**Sensordatenvorverarbeitung** Die empfangenen oder gemessenen Rohsignale der z. B. im vorherigen Abschnitt geschilderten Sensoren enthalten dabei neben dem Nutzsignal auch Störsignale (Rauschen). Die Rohsignale (z. B. eine Spannung) werden dabei als physikalische Messgrößen interpretiert, die schließlich die Rohdaten des Sensors bilden. In dem Schritt der Vorverarbeitung wird dabei versucht, das Nutzsignal aus den Rohdaten zu extrahieren.

**Signalverarbeitung und Merkmalsextraktion** Im Rahmen der Signalverarbeitung und Merkmalsextraktion werden Informationen aus dem Umfeld des Fahrzeugs über die z. B. im vorherigen Abschnitt geschilderten Sensoren erfasst. Das entspricht dem eigentlichen Messen. Die empfangenen oder gemessenen Rohsignale enthalten dabei neben dem Nutzsignal auch Störsignale (Rauschen). Die Rohsignale (z. B. eine Spannung) werden dabei als physikalische Messgrößen interpretiert (z. B. eine Intensität), die schließlich die Rohdaten des Sensors bilden. In dem Schritt, der als Wahrnehmung bezeichnet wird, werden mittels Annahmen bzw. Heuristik Merkmale extrahiert, z. B. Kanten oder Extremwerte, aus denen Merkmalshypothesen und dann Objekthypothesen eines angenommenen Objekts abgeleitet werden. [6]

**Datenassoziation** Die zuvor gewonnenen Merkmalshypothesen werden im Prozess der Datenassoziation den bereits im System vorhandenen Objekthypothesen zugeordnet. Dabei hat die Datenassoziation einen entscheidenden Einfluss auf die Qualität der Schätzung. Im Fall einer falschen Zuordnung kann es zu einem Informationsverlust oder einer Fehlinformation kommen. Dementsprechend müssen Zuordnungshypothesen gebildet werden, wobei sich dieses Vorgehen in zwei Teilschritte unterteilen lässt. Zum einen in die Aufstellung der möglichen Zuordnungshypothesen und die Auswahl der prinzipiell möglichen Hypothesen. Dabei können verschiedene Methoden herangezogen werden, um Zuordnungshypothesen aufzustellen (physikalisches Modell, Szenenwissen, probabilistische Modelle und weitere). Eine Detaillierung der Methoden ist in [12] aufgelistet.

**Klassifikation** Aufgrund zugeordneter Eigenschaften werden Objekthypothesen innerhalb der Klassifikation einer vordefinierten Klasse zugeordnet. Dies kann auf Basis der Rohdaten des Sensors, aber auch aus den geschätzten Zustandsvariablen der Objekthypothese gewonnen werden.

### 2.2.2 Planung – Situationsanalyse

Als Planung oder auch als Situationsanalyse wird im Kontext der Fahrerassistenzsysteme verstanden, anhand der vorhandenen Aufnahmen des Umfelds über die Sensorik und der Verarbeitung dieser, eine Entscheidung über mögliche Aktionen zu treffen. Dabei sind die Algorithmen dahinter so vielfältig wie die Assistenzsysteme selbst. So kann je nach Assistenzsystem eine Objektvorhersage erforderlich sein, die beispielsweise auf Grundlage der Umfelddaten eines Objekterkennungsalgorithmus (über Histogramme und Merkmalsvektoren) ein Objekt detektiert, um so mittels eines Trackingalgorithmus (Kalman-Filter) die Position eines Objekts vorhersagen zu können. In der Konsequenz muss dann innerhalb der Situationsanalyse aus der Eigenbewegung des Fahrzeugs und der voraussichtlichen Bewegung des Objekts eine Entscheidung getroffen werden, die u. a. zu einem Regeleingriff des Systems führt.

Die Situationsanalyse stellt somit das Verbindungsstück zwischen der Umfelddatenverarbeitung und der eigentlichen Assistenzfunktion dar. Am Beispiel des ACC kann im Hintergrund des Systems ein Zustandsautomat implementiert sein, der das Verhalten der Anwendung in verschiedenen Situationen festlegt. Als Stichpunkte werden die verschiedenen Planungskomponenten aufgeführt. Weiterführende Informationen finden sich in [6].

- Pfadplanung – plant eine Route, um kollisionsfrei in einer bekannten Karte von einem Start zu einem Zielpunkt zu kommen.
- Objektvorhersage – zur Detektion von Verkehrsteilnehmern im Fahrzeugumfeld, hier stellt das Objekt-Tracking eine Schlüsselkomponente dar.
- Verhaltensplaner – ist das zentrale Element in einem Fahrerassistenzsystem, um zu entscheiden, was der nächste Schritt ist.

- Trajektorienplaner – funktioniert ähnlich zur Pfadplanung, setzt dabei den innerhalb der Pfadplanung geplanten Pfad akzeptabel um (ohne Komfort und Sicherheitseinbußen).

### 2.2.3 Steuerung – Reaktion

Die Steuerung umfasst die eigentliche Reaktion des Fahrerassistenzsystems, die sich in einem Beschleunigen, Bremsen oder Steuern des Fahrzeugs äußert. Diese Reaktionen sind das Ergebnis der Planungsphase.

Für die aktive Ausführung von Systemeingriffen werden Aktuator-Systeme benötigt, um das Fahrzeug zu steuern. Hierbei kommen klassische Verfahren der Regelungstechnik zum Einsatz. Der Vollständigkeit halber seien auch Systeme wie das ESP und Electric Power Steering (EPS) erwähnt. Dabei ermöglichen die Aktoren im ESP einen Längsdynamikeingriff in Form einer Beschleunigung oder Verzögerung. Dahingegen kann mittels des EPS ein Querdynamikeingriff vorgenommen werden. Nähere Erläuterungen zu den Systemen und alternativen Aktuatoren können dem Standardwerk [6] entnommen werden.

### 2.2.4 Auswahl von Fahrerassistenzsystemen

Nachdem in den vorangegangenen Kapiteln bereits verschiedene Assistenzsysteme erwähnt wurden, sollen in diesen Abschnitt verschiedene Fahrerassistenzsysteme vorgestellt werden, die u. a. für die Umsetzung einer „Follow-Me“-Funktion erforderlich sind.

#### 2.2.4.1 Längsführungsassistenz

Unter den Längsführungsassistenten fallen mehrere Assistenten, wie ein Tempomat, Limiter oder ein ACC-System. Wobei hier der Fokus auf der höchsten Ausbaustufe, dem ACC liegt. Das ACC auch als Abstandsregeltempomat, aktive Geschwindigkeitsregelung, automatische Distanzreglung, Automatic Cruise Control, Autonomous Intelligent Cruise Control oder Distronic bekannt, zeichnet sich durch eine sich an die Verkehrssituation anpassende Fahrgeschwindigkeit aus. Die Anforderungen an das System wird u. a. in der ISO 15622 (Intelligent transport systems - adaptive cruise control systems - Performance requirements and test procedures) beschrieben. [13]

Der Begriff ACC leitet sich von der in Nordamerika und Japan bekannten Bezeichnung Cruise Control (CC) ab. In Deutschland entspricht das dem einfachen Tempomaten, der als Teilfunktion, das Regeln auf eine von einer fahrzeugführenden Person gesetzte Wunschgeschwindigkeit, im ACC enthalten ist. Die Hauptfunktion des ACC bezieht sich auf die Anpassung der Fahrzeuggeschwindigkeit an die Geschwindigkeit des direkt vorausfahrenden Fahrzeugs. Das System funktioniert dabei so, dass im Fall eines vorausfahrenden, langsameren Fahrzeugs als das eigene, mit der Wunschgeschwindigkeit fahrende Fahrzeug, das ACC als Regelaufgabe das eigene Fahrzeug verzögert, um einen von der fahrzeugführenden Person eingestellten Sekundenabstand nicht zu unterschreiten und einzuhalten. Dabei wird auf eine Zeitlücke als Abstandsmaß zurückgegriffen, anstatt auf einen räumlichen Bezug. Das rührt daher, dass für die Regelung oder alternativ den Fahrereingriff eine Reaktionszeit benötigt wird, die unabhängig von der Geschwindigkeit ist. Nach dem Zusammenhang von, $v = s/t$ genauer gesagt $s = v \cdot t$, ergibt sich ein relativer Abstand, der abhängig von der Fahrzeuggeschwindigkeit und dem, von der fahrzeugführenden Person gewählten, Sekundenabstand $t$ ist. Verlässt das vorausfahrende Fahrzeug den Fahrkorridor und kein anderes Zielobjekt fährt in den Fahrkorridor hinein, nimmt das ACC ohne eine weitere Aktion der fahrzeugführenden Person die Regelung zur Wunschgeschwindigkeit wieder auf. [6]

**Systemanforderungen nach ISO 15622 in Bezug auf den Fahrzustand [13]**

Freifahrt: Konstante Geschwindigkeitsreglung mit hohem Regelkomfort und geringer Abweichung der Setzgeschwindigkeit sowie Bremseingriff bei reduzierter Wunschgeschwindigkeit, bis diese erreicht ist, oder im Fall einer Gefällefahrt.

Folgefahrt: Übernahme der Geschwindigkeit des vorausfahrenden Fahrzeugs mit Schwingungsdämpfung, um Geschwindigkeitswechsel des Zielfahrzeugs nicht mitzuübernehmen. Einreglung des Sekundenabstands auf die gewählte Zeitlücke, dabei soll eine Orientierung an das normale Verhalten erfolgen, wenn es zu einer Abstandsverkürzung durch einen Einscherer kommt. Dabei hat die Regelung mit der von der fahrzeugführenden Person erwartenden Dynamik zu erfolgen. Gleichzeitig muss eine Kolonnenstabilität für den Fall des Folgens anderer ACC-Fahrzeuge gewährleistet sein, während ausreichende Beschleunigungsfähigkeit für ein zügiges Mitschwimmen und Aufschließen erforderlich ist. Das System muss eine gute Verzögerungsfähigkeit in ca. 90 % des fließenden Verkehrs realisieren können. Weiterhin muss eine automatische Zielobjekterkennung bei Annäherung oder Ein- und Ausscheren vorausfahrender Fahrzeuge gegeben sein. Im Fall eines kompletten Geschwindigkeitsbereichs abdeckenden (engl. FullRange) ACC ist eine Regelung bereits ab 0 km/h, insbesondere im Kriechbereich erforderlich.

Bei Annäherung: Im Fall einer langsamen Annäherung muss zügig zum Soll-Abstand geregelt werden. Besteht eine zügige Annäherung auf ein Zielobjekt, muss ein für die fahrzeugführende Person vorhersehbarer Verzögerungsverlauf gegeben sein, um einer Unsicherheit der fahrzeugführenden Person bezüglich einer Notwendigkeit eines Eingriffs vorzubeugen. Kommt es zu einer Unterschreitung des Sekundenabstands, muss ein für die fahrzeugführende Person typisches „Zurückfallen“ des Fahrzeugs umgesetzt werden.

Beim Anhalten: Das System muss auf einen sinnvollen Halteabstand einregeln. Ein sicheres Halten im Stand mittels Betriebsbremse muss gewährleistet sein. Kommt es ohne einen Fahrereingriff zu einer Systemabschaltung im Stillstand, ist das Fahrzeug in einen sicheren Haltezustand ohne Hilfsenergie zu versetzen.

Funktionsgrenzen: Minimale Soll-Geschwindigkeit > 30 km/h. Die Zeitlücke darf im eingeschwungenen Systemzustand den Wert von 1 s nicht unterschreiten. Der Fahrer hat immer Priorität und kann das System z. B. durch Betätigung der Bremse oder einer Taste deaktivieren bzw. durch eine Fahrpedalbetätigung übersteuern. Die Beschleunigungen müssen innerhalb der Grenzen von $a_{min} = -3,5 m/s^2$ bis $a_{max} = 2,5 m/s^2$ liegen, dabei gibt es weitere Grenzen für die Beschleunigung in Abhängigkeit der Fahrzeuggeschwindigkeit.

**Sensorik** Für die Umsetzung des ACC gibt es verschiedene Ansätze. Für die Abstands- und Geschwindigkeitsmessung wird meist ein Radar verwendet. Zur Lokalisierung der Objekte wird aktuell ein Lidar-Sensor oder Kameratechnik genutzt.

#### 2.2.4.2 Querführungsassistenz

Unter diesen Assistenzsystemen laufen Systeme, die den Fahrer darin unterstützen, das Fahrzeug auf der Fahrbahn zu halten. Die Systeme lassen sich aus technischer Sicht in zwei Gruppen unterteilen. Erstens in die Gruppe der Spurverlassenwarner (engl. Lane-Departure-Warning), die im Fall eines Spurverlassens rein visuell, akustisch oder haptisch unterstützen. Zweitens in die Gruppe der Spurhalteassistenten (engl. Lane-Keeping-Assistance), die das Fahrzeug durch einen aktiven Lenkeingriff in der Fahrspur halten. Dabei kann nochmals unterschieden werden in Systeme, die

das Fahrzeug vor dem Verlassen der Fahrspur schützen und in Systeme, die den Fahrer in der Querführung so unterstützen, dass das Fahrzeug in der Fahrbahnmitte durch aktive Lenkeingriffe verbleibt.

Aus Unfallstatistiken geht hervor, dass mehr als ein Drittel aller Unfälle auf ein Abkommen von der Fahrbahn zurückzuführen ist. Dies verdeutlicht, dass durch Querführungsassistenzen neben dem Fahrkomfort auch die Sicherheit gesteigert werden kann. [6]

**Systemanforderungen** Als Basis für diese Assistenzfunktion muss sich die Fahrzeugposition relativ zur Grenze der Fahrbahn bestimmen lassen. Dies erfolgt meist durch eine Fahrbahnlinie. Wobei mindestens eine Linie (engl. „Single Line Detection“) erkannt werden muss, für die Systeme, die die fahrzeugführende Person lediglich informieren. Um ein Fahrzeug aktiv in der Spurmitte halten zu können, ist eine Ermittlung der Fahrzeugposition relativ zur Mitte des Fahrstreifens erforderlich. Ferner muss der zukünftige Streckenverlauf bekannt sein, damit Systeme wie der Spurhalteassistent funktionieren können.

Um diese Systemanforderungen zu erfüllen, sind Umfeldsensoren mit einer hohen Genauigkeit erforderlich, so erfolgt z. B. die Erkennung der Fahrstreifenmitte anhand der rechten und linken Fahrstreifenmarkierung, die vom System erkannt werden müssen (engl. „Dual Line Detection“). Dabei sollte die Erkennung auf möglichst allen Straßen und auch bei widrigen Umwelteinflüssen möglich sein. Gleichzeitig dürfen die Systeme bei gewolltem Spurwechsel, z. B. bei einem Überholvorgang, diesen durch Fehlinformationen oder Fehleingriffe nicht behindern. Beim Halten oder während der Rückführung des Fahrzeugs in die Fahrspur ist ein aktiver Systemeingriff in die Fahrzeugquerführung erforderlich. Der Systemeingriff muss dabei immer von der fahrzeugführenden Person überstimmbar sein. Des Weiteren darf das System beim Halten des Fahrzeugs innerhalb der Fahrspur keine hochfrequente ständige Lenkbewegung ausführen, sondern muss ein natürliches Lenkverhalten abbilden. Der aktuelle Systemzustand und das Verhalten sind der fahrzeugführenden Person eindeutig anzuzeigen.

In den aktuellen Ausbaustufen des Querführungs-Assistenten hat die fahrzeugführende Person bisher noch die Verantwortung für die Fahrzeugquerführung. Aktuelle Entwicklungen zeigen jedoch, dass bereits erste Systeme auf den Markt kommen, bei denen die fahrzeugführende Person von der Aufgabe der Querführung in bestimmten Systemgrenzen (z. B. Autobahnfahrt, mit Geschwindigkeiten bis 60 km/h) entbunden wird, diese jedoch auf Systemverlangen wieder übernehmen muss. Für die Spurverlassenwarner gibt es eine Norm in der ISO 17361:2017. [14] Ebenso gibt es für den Spurhalteassistenten eine zu berücksichtigende Norm ISO 11270:2014. [15] Dabei werden u. a. Anforderungen an die maximale Querbeschleunigung von 3 m/s² durch einen aktiven Lenkeingriff definiert. Infolge eines Querrucks dürfen die Querbeschleunigungen nicht mehr als 5 m/s² betragen.

**Sensorik** Durchgesetzt haben sich Kamerasysteme (monokulare), die hinter der Windschutzscheibe in der Spiegeleinheit verbaut sind. Dieses System ist in der Lage, Fahrstreifenbegrenzungen zu detektieren. Dabei liegt in der hohen Auflösung und dem großen Sichtbereich ein Vorteil der Kamerasysteme. Kommen Farbkameras zum Einsatz, so sind auch Sondersituationen wie eine Baustellenfahrt auflösbar, sodass die Kamera zwischen weißen und gelben Linien unterscheiden kann. Als weiter Ergänzung werden aktuell u. a. Lidar-Systeme eingesetzt, um z. B. Bordsteinkanten oder Leitplanken innerhalb von Baustellen als Fahrbahnmarkierung zu erkennen.

#### 2.2.4.3 Kombinierte Längs- und Querführungsassistenz – „Follow-Me“

Eine Kombination aus einem Abstandsregeltempomat mit gleichzeitiger Nutzung des Spurhalteassistenten findet immer mehr Einzug in die Fahrzeuge. Lange Zeit war der rechtliche Rahmen hierfür nicht gegeben, weswegen die Systeme trotz technischer Umsetzbarkeit bisher nicht zum Einsatz kamen. Die Hersteller betiteln den Verbund aus den beiden Systemen dabei unterschiedlich, z. B. Travel Assist (VW) [16], Stauassistent (Audi) oder der Staupilot / Drive Pilot (Mercedes). [17] Dabei unterscheiden sich die Systeme sowohl in ihrem Automatisierungsgrad (SAE-Level 2 bis 3) als auch in den Systemgrenzen (0–210 km/h oder 0–60 km/h).

Besonders hervorgehoben sei an dieser Stelle der Mercedes-Benz Drive Pilot, das als erstes Serienfahrzeug ein SAE-Level 3 System bis 60 km/h implementiert hat. Bei aktiviertem Drive Pilot übernimmt das Fahrzeug die Längs- und Querführungsaufgabe bis 60 km/h, sodass die fahrzeugführende Person sich Nebentätigkeiten widmen kann, wobei sie das Fahrzeug aber nach Systemaufforderung wieder steuern muss. Als Sensorik dienen dabei eine Vielzahl von Sensorsystemen wie Radar, Lidar und Kamera sowie Ultraschallsensoren. [17]

Die im Rahmen dieser Arbeit zu implementierende „Follow-Me“-Funktion entspricht dabei der Funktionalität des Drive Pilots. Die Aufgabe zielt dabei auf eine selbstständige Längs- und Querführungsfunktion ab, die an einem JetRacer umgesetzt werden soll. Der JetRacer soll auch einem vorausfahrenden JetRacer folgen können. Als Unterschied sei bereits an dieser Stelle erwähnt, dass, im Gegensatz zu den Sensorsystemen des Mercedes, ausschließlich ein Kamerasystem als Sensorik genutzt wird. Gleichzeitig gelten an Sicherheit und Robustheit sowie Komfort andere Systemanforderungen, da es sich bei dem JetRacer um eine Art Modellfahrzeug handelt und nicht um ein Fahrzeug, das am realen Straßenverkehr teilnimmt.

In diesem Abschnitt 2.2 wurde ein Einblick in die für konventionelle Fahrerassistenzsysteme erforderlichen Komponenten zur Gestaltung ihrer Systemfunktionalität bis hin zur Vorstellung einer kleinen Auswahl von Fahrerassistenzsystem gegeben. Dabei wurde nicht auf eine mögliche Unterstützung von künstlichen Intelligenz-Systemen eingegangen, dies ist dem nachfolgenden Abschnitt vorbehalten.

Nachdem das Fahrzeug seine Umgebung mithilfe von Sensorik, Fusion, Objektklassifizierung und Tracking wahrnehmen und entsprechend der aktuellen Umfeldsituation ausgewählte Applikationen ausführen kann, gilt es den Einfluss eines KI-gestützten Systems darzulegen. Im Rahmen dieser Arbeit soll die „Follow-Me“-Funktion mittels eines KI-gestützten Systems abgebildet werden. Im nächsten Abschnitt wird geklärt, wie eine KI bestehende Fahrerassistenzsysteme stärken kann.

## 2.3 KI-Unterstützung bei Fahrerassistenzsystemen

In den vorangestellten Ausführungen wurde zur Implementierung der Fahrerassistenzfunktionen eine Unterteilung des Problems in,

- Wahrnehmung (Messen und Erkennen),
- Planung (Situationsanalyse) und
- Steuerung

zugrunde gelegt. Dabei werden bei konventionellen Systemen innerhalb der Bestandteile der Assistenzsysteme feste Algorithmen implementiert, die z. B. auf Grundlage von Sensordaten Entscheidungen treffen.

Beispielsweise erfolgt für einen Spurhalteassistenten die Objekt- bzw. Spurerkennung auf Basis der Rohdaten des Sensors, mittels fester Heuristiken, woraus Merkmale extrahiert werden, aus denen eine Objekthypothese und später ein Objekt abgeleitet wird.

Bereits an dieser Stelle kommen mitunter Algorithmen wie der Histogram of Oriented Gradients (HOG) Algorithmus zum Tragen, der anhand der in Abbildung 2.4 dargestellten Einordnung bereits als Teil des „maschinellen Lernen“ gesehen werden kann und somit per Einordnung als künstliche Intelligenz (KI) zählt.

Im Rahmen dieser Arbeit wird jedoch unter KI-Unterstützung der Bereich des Deep Learning unter Verwendung von großen Künstlichen-Neuronalen-Netzen (KNN) verstanden und nach aktuellen Recherchen nur bei dem Automobilhersteller Tesla [18] zum Einsatz kommen. Mittels dieser Techniken werden komplexe mathematische Algorithmen durch komplexe neuronale Netze ersetzt, die nach den vorgegebenen Datensets entsprechend zu guten bis sehr guten Ergebnissen kommen. Aktuell erreicht der Objekterkennungsalgorithmus Contrastive Captioners - Objekterkennung Algorithmus (CoCa) eine Genauigkeit von 91 % in der Anwendung auf den ImageNet Datensatz, der aus 14.197.122 klassifizierten Bildern besteht. [19]

Die Vorteile, die sich in der Anwendung solcher Deep Learning Algorithmen zeigen, sind, dass sie bei großen Datensätzen wesentlich schneller und dabei genauer sind als bspw. die Algorithmen des Maschinelles Lernen (engl. Machine Learning) (ML) Anwendung von Support Vector Machine (SVM). Dies ist notwendig, wenn sich das Fahrzeug schnell bewegt und die Situation in Echtzeit analysiert werden muss, um z. B. einen autonomen Spurwechsel zu veranlassen, damit einer Gefahr ausgewichen wird. Ein Nachteil dieser Deep Learning Algorithmen ist dabei, dass sie eine sehr hohe Rechenleistung sowie große Datensätze benötigen. [20]

Schlussfolgernd kann gesagt werden, dass auch heute schon definitionsgemäß eine KI innerhalb von Fahrzeugen eingesetzt wird. Meist im Bereich des maschinellen Lernens zur Objekterkennung, die u. a. auf ML-Algorithmen wie SVM, Bayes oder Entscheidungsbäume zurückzuführen ist. Dabei sind diese Methoden nicht so leistungsfähig, wie die Nutzung von neuronalen Netzen, die auch innerhalb dieser Arbeit betrachtet werden sollen. [20]

Eine Einführung in die Grundbegriffe der KI sowie ein Überblick über die Begrifflichkeiten soll im nächsten Kapitel vermittelt werden.

## 2.4 Grundlagen KI und neuronale Netze

Um die Zusammenhänge und die Möglichkeiten von KI-Systemen zu verstehen (engl. Artificial Intelligence (AI)), wird in diesem Kapitel erläutert, was ein KI-System ist und welche Formen es gibt.

Der Begriff KI geht zurück auf das Jahr 1956, in dem, im Rahmen einer Konferenz von Wissenschaftlern, der Begriff nicht nur definiert, sondern auch eine wissenschaftliche Disziplin begründet wurde. Jedoch stellt sich die Frage, was genau KI ist. Eine präzise Antwort gibt es nicht. Im Allgemeinen wird dann von einer KI gesprochen, wenn ein Computersystem eine Aufgabe übernehmen kann, dabei für die Bewältigung der Aufgabe aber nicht jeder Schritt eigens programmiert werden muss, wohingegen die Lösung der Aufgabe von einem Menschen Intelligenz verlangen würde. [21]

### 2.4.1 Einordnung der Begriffe

Mit der Bezeichnung KI werden auch oft Begriffe, wie maschinellen Lernens oder Deep Learning (mehrschichtiges Lernen) in Verbindung gebracht. Um Verwechslungen der Begrifflichkeiten vorzubeugen, werden die drei Begriffe anhand der Abbildung 2.4 erläutert.

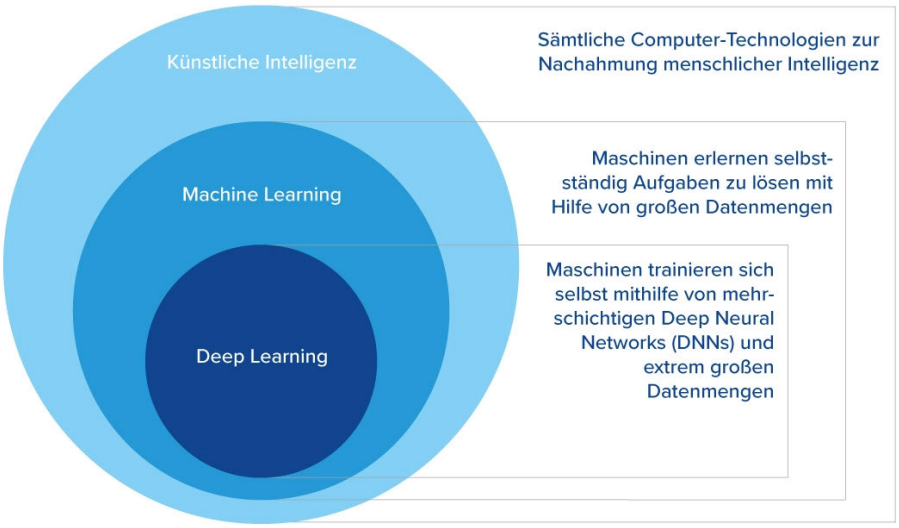


**Abbildung 2.4:** Formen der Künstlichen Intelligenz [22]

Darauf aufbauend ist KI ein Verfahren, mit dem Computer menschliche Intelligenz imitieren können. Enthalten ist auch das maschinelle Lernen, wie anhand der Abbildung 2.4 zu sehen. Das ML ist ein Teilbereich der KI, der Techniken (z. B. Deep Learning) umfasst, die Computer durch Sammeln von Erfahrungen beim Lösen von Aufgaben verbessern können. Deep Learning ist wiederum ein Teilbereich des ML und basiert auf künstlichen, neuronalen Netzen. Der Lernprozess wird als Deep Learning bezeichnet, da die Struktur von künstlichen neuronalen Netzen aus einer Eingabe- und einer Ausgabeschicht sowie mehreren verborgenen Schichten besteht, siehe dazu auch Abbildung 2.6. Aufgrund dieser Struktur kann ein Computer durch seine eigene Datenverarbeitung lernen.

### 2.4.2 Lernarten des maschinellen Lernens

Innerhalb des ML können vier grundsätzliche Lernarten unterschieden werden, siehe hierzu auch Abbildung 2.5. Angelehnt an dem menschlichen Lernen, lernt auch ein KI-System aus Beispielen, um in Zukunft besser zu agieren. Dafür ist ein Trainingsprozess notwendig, der auf verschiedenen Methodiken basiert, die nachfolgenden erläutert werden.

#### 2.4.2.1 Überwachtes Lernen

Bei dieser maschinellen Lern-Variante (engl. Supervised Learning, siehe Abbildung 2.5 – blauer Bereich) wird das Modell mit Daten trainiert, bei denen die Zusammenhänge bekannt sind und Werte für die zu beschreibende Größe vorliegen. Anschließend können die trainierten Modelle auf Fälle angewandt werden, in denen die Werte der zu beschreibenden Variable unbekannt sind. Das Ziel ist ein Modell, das fallweise die Ausprägung einer zu beschreibenden Größe anhand von mehreren beschreibenden Variablen vorhersagt.

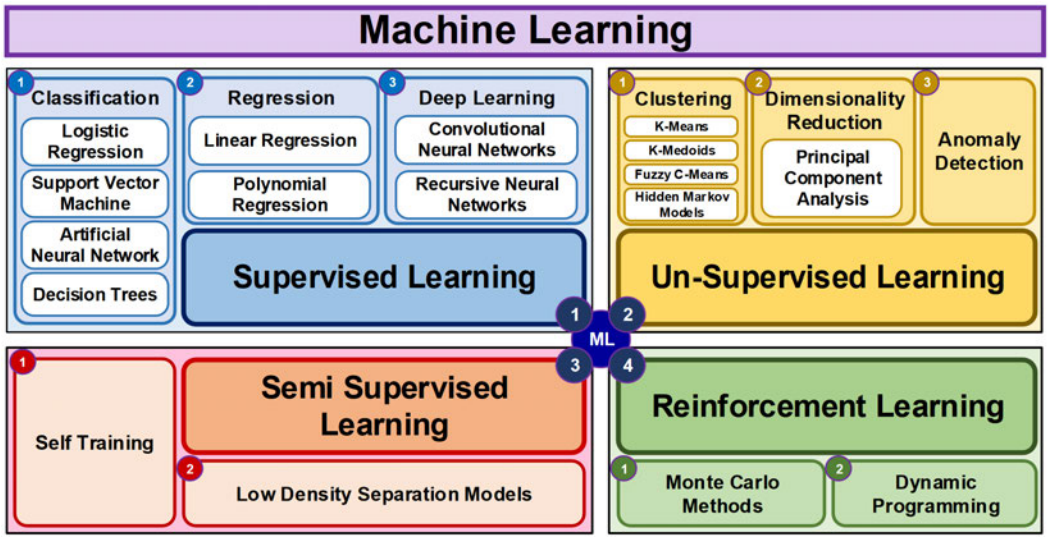


**Abbildung 2.5:** Unterkategorien des maschinellen Lernens [23]

Zum Beispiel wird ein Bild mit einem Stoppschild dem KI-System als Eingangsdaten zur Verfügung gestellt, verknüpft mit der Information, dass es sich bei diesem Bild um ein Stoppschild handelt. Diese Abfolge wird mit verschiedenen Ansichten von Stoppschildern mehrfach wiederholt, sodass das KI-System anhand von immer mehr Beispielen weiß, wie ein Stoppschild aussieht, bis es ein vorher noch nicht gezeigtes Bild eines Stoppschilds zuverlässig erkennen kann. [24]

Als Vorhersagemethoden im überwachten Lernen kommen die Klassifikations- und Regressionsverfahren zum Einsatz. Eine Klassifikation liegt dann vor, wenn die zu beschreibende Variable kategorial ist. Eine Regression hingegen sagt die Werte kontinuierlicher Variablen vorher.

#### 2.4.2.2 Unüberwachtes Lernen

Dem überwachten Lernen steht das unüberwachte Lernen (engl. Unsupervised Learning, siehe Abbildung 2.5 – gelber Bereich) gegenüber. Hierbei kann das KI-System nicht auf Beispieldaten zurückgreifen, sondern soll vielmehr selbstständig versteckte Gruppen und Muster aus einer großen Datenmenge erkennen, ohne weitere Erläuterung durch einen Menschen.

Der grundsätzliche Unterschied zum überwachten Lernen wird dadurch deutlich, dass das unüberwachte Lernen nicht dafür ausgelegt ist, eine Vorhersage für eine bekannte Zielvariable zu berechnen. Es kann also z. B. nicht sagen, ob es sich um ein Stoppschild handelt. Das unüberwachte Lernen eignet sich auf der anderen Seite aber hervorragend zum Clustern unbekannter großer Datenmengen. [24]

#### 2.4.2.3 Teil-überwachtes Lernen

Eine Zwischenform zu diesen beiden Lernarten, des unüberwachten und überwachten Lernens, stellt das teil-überwachte Lernen (Semi-supervised Machine Learning, siehe Abbildung 2.5 – roter Bereich) dar, das sowohl Beispieldaten mit genauen Zielvariablen als auch unbekannte Daten analysieren kann. Die Einsatzgebiete sind dabei identisch mit denen des überwachten Lernens. Der größte Unterschied besteht darin, dass in dem Lernprozess nur eine geringe Menge an Daten mit bekannter Zielvariable genutzt wird, um eine große Datenmenge zu bewerten, bei der die Zielvariable noch nicht bekannt ist. Das hat den Vorteil, dass schon mit einer geringen Menge von bekannten

gekennzeichneten bzw. gelabelten Daten trainiert werden kann. Dieses Vorgehen findet häufig Anwendung, da bereits die Beschaffung von bekannten Beispieldaten aufwendig und kostenintensiv ist, da häufig Menschen diese Daten manuell aufbereiten. Ein Beispiel hierfür ist, das Kennzeichnen (engl. labeling) von Bildern. [25]

#### 2.4.2.4 Verstärktes Lernen

Eine weitere Lernmethode ist das verstärkte Lernen (Reinforcement Learning, siehe Abbildung 2.5 – roter Bereich), bei dem das KI-System „belohnt“ oder durch eine Kostenfunktion bewertet wird, wenn eine Aufgabe richtig gelöst wurde, z. B. ein Auto von einer Straße unterschieden werden konnte. Das Ziel ist es, dass das System, unter der Vorgabe, die Belohnung zu maximieren, selbstständig eine Strategie zur Lösung des Problems erlernt. [26]

### 2.4.3 Künstliche neuronale Netze

Eine wichtige und in dieser Arbeit verfolgte Anwendung des überwachten ML sind KNN (engl. Artificial Neural Networks (ANN)). Diese bilden u. a. die Grundlage für die KI.

KNN sind Algorithmen, die dem menschlichen Gehirn nachempfunden sind, und finden Anwendung im ML, aber auch im Deep Learning Bereich. Somit lassen sich verschiedene Datenquellen wie Geräusche, Texte, Tabellen oder Bilder interpretieren und Informationen bzw. Muster aus den Daten extrahieren, um diese auf unbekannte Daten anzuwenden. Das ermöglicht eine, durch die Daten getriebene Vorhersage, deren Qualität von den Eingangsdaten, aber auch dem KNN abhängt. KNN können unterschiedlich komplex aufgebaut sein, weisen aber grundsätzliche Strukturen von gerichteten Graphen auf.

KNN bestehen vereinfacht dargelegt aus Knoten, auch Neuronen genannt, die Informationen von außen oder anderen Neuronen aufnehmen, bearbeiten und als Ergebnis ausgeben. Die Verarbeitung erfolgt dabei über drei verschiedene Schichten, denen jeweils Neuronen zugeordnet sind. KNN bestehen demnach aus

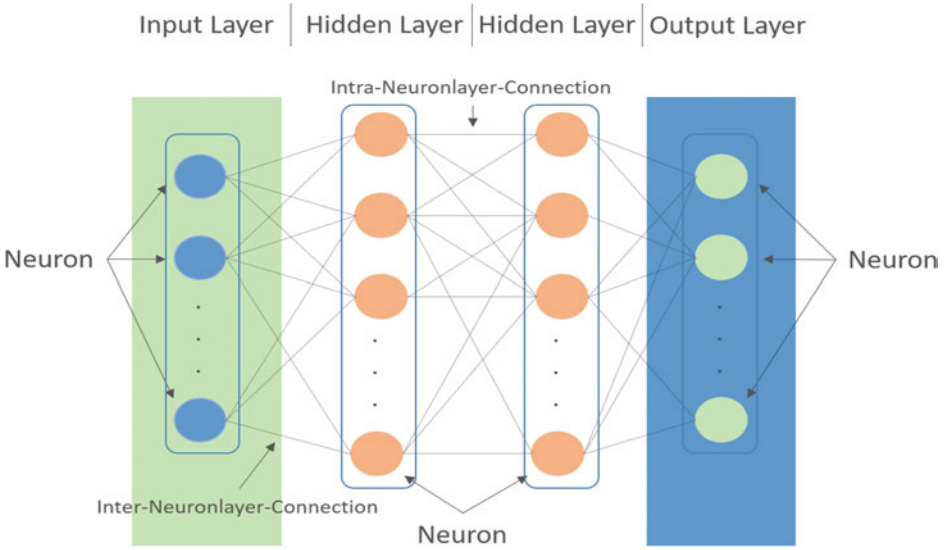


**Abbildung 2.6:** Schichten eines künstlichen neuronalen Netzes [24]

- Eingabeschicht (engl. Input Layer),
- Zwischenschicht(en) oder verborgene Schicht (engl. Hidden Layer) und
- Ausgabeschicht (engl. Output Layer).

**Eingabeschicht** Diese Schicht bildet den Startpunkt des Informationsflusses in einem KNN. Eingangssignale bzw. Informationen werden durch Input-Neuronen zu Beginn dieser Schicht aufgenommen und am Ende gewichtet an die Neuronen der ersten Zwischenschicht weitergegeben. Dabei gibt ein Neuron der Eingabeschicht die jeweilige Information an alle Neuronen der ersten Zwischenschicht weiter.

**Zwischenschicht(en)** Die Neuronen der Zwischenschicht(en), die verborgenen Neuronen (engl. hidden neurons), bilden innere Informationsmuster ab. Zwischen der Eingabe- und der Ausgabeschicht befindet sich in jedem KNN mindestens eine Zwischenschicht, auch Aktivitätsschicht oder verborgene Schicht genannt. Je nach Art des KNN kann dieses eine unterschiedliche Anzahl von verborgenen Schichten ausweisen. Hat ein KNN besonders viele Zwischenschichten, wird dieses auch als tiefes neuronales Netz (engl. deep neural network) bezeichnet, die im Bereich des Deep Learning untersucht werden. Theoretisch ist die Anzahl der möglichen verborgenen Schichten in einem KNN unbegrenzt. Jedoch bewirkt jede hinzukommende verborgene Schicht auch einen Anstieg der benötigten Rechenleistung, die für den Betrieb des Netzes notwendig ist, wodurch die Tiefe der Netze durch die verfügbare Rechenleistung begrenzt ist.

**Ausgabeschicht** Die Ausgabeschicht liegt hinter den Zwischenschichten und bildet die letzte Schicht eines KNN. In der Ausgabeschicht angeordnete Ausgabe-Neuronen (engl. output neurons) sind jeweils mit allen Neuronen der letzten Zwischenschicht verbunden. Die Ausgabeschicht stellt den Endpunkt des Informationsflusses in einem KNN dar und enthält das Ergebnis der Informationsverarbeitung durch das Modell.

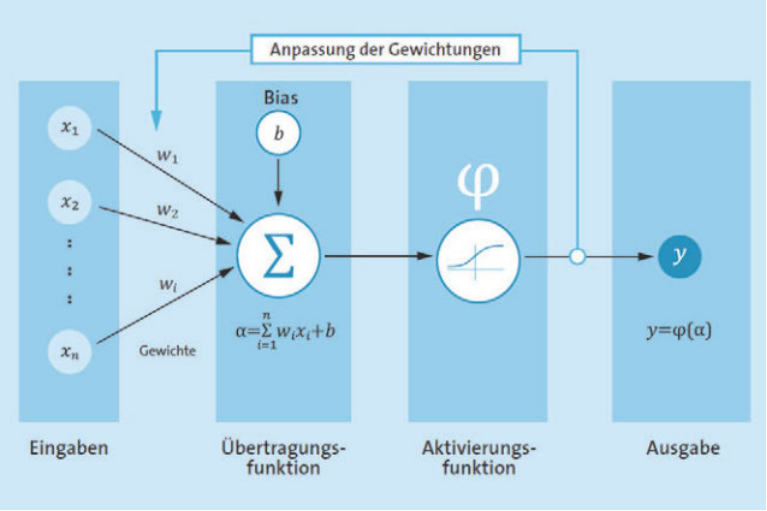


**Abbildung 2.7:** Darstellung eines künstlichen Neurons [27]

Wie bereits beschrieben sind die Neuronen zur Informationsweitergabe miteinander verbunden, siehe Abbildung 2.7. Diese Verbindung wird auch Kante genannt und entspricht in der Biologie der Synapse. Dabei beschreiben Gewichte, gemäß Abbildung 2.7, die Intensität / Stärke des Informationsflusses entlang der Verbindung bzw. Kante. Dementsprechend gibt jedes Neuron, entsprechend dem eigenen Gewicht, die durchfließende Information gewichtet weiter. Die Übertragungsfunktion berechnet die Anregung des Neurons anhand der Summen-Berechnung der gewichteten Eingaben. Dabei besteht für jedes Neuron noch die Möglichkeit, die Information mittels eines Schwellenwertes (engl. bias) zu verzerren.

Das Ergebnis der Gewichtung und Verzerrung wird, bevor es an die Neuronen der nächsten Schicht weitergeleitet wird, oft durch eine Aktivierungsfunktion geleitet, um einen bestimmten Wertebereich des Outputs zu erzwingen. Das Ergebnis der Aktivierungsfunktion, die z. B. eine Sigmoid, eine Rectified Linear Unit oder Heaviside Funktion o. ä. sein kann, wird dann an die Neuronen der nächsten Schicht weitergeleitet. [28]

Ein wichtiges Merkmal von Systemen mit KI ist die Fähigkeit, selbstständig zu lernen. Anders als bei klassischer Software, die Probleme und Fragen auf Basis von vorher festgelegten Regeln abarbeitet, können selbstlernende ML-Algorithmen die besten Regeln für die Lösung bestimmter Aufgaben selbst erlernen.

KNN lernen dabei wie folgt: Nach dem Aufbau der Struktur des KNN erhält jedes Neuron ein zufälliges Anfangsgewicht. Anschließend werden die Eingangsdaten in das Netz gegeben und jedes Neuron gewichtet die Eingangssignale anhand seines Gewichts, Schwellwertes und der Aktivierungsfunktion und gibt das Ergebnis an die Neuronen der nächsten Schicht weiter. An der Ausgabeschicht wird dann das Gesamtergebnis berechnet. Im ersten (Lern-) Schritt wird dieses Ergebnis aufgrund der Tatsache, dass alle Neuronen ein zufälliges Anfangsgewicht erhalten haben, in der Regel wenig mit dem bekannten tatsächlichen Ergebnis zu tun haben. Es ist jedoch möglich, die Größe des Fehlers zu berechnen und den Anteil, den jedes Neuron an diesem Fehler hatte, sowie das Gewicht jedes Neurons ein wenig in die Richtung zu verändern, die den Fehler minimiert. Dann erfolgt der nächste Durchlauf, eine erneute Messung des Fehlers und Anpassung der Gewichte und so weiter. Durch einen solchen Prozess lernt das KNN zunehmend besser, von den Eingabedaten auf die bekannten Ausgabedaten zu schließen. [28]

Im Rahmen des Unterabschnitt 4.1.2 wird näher auf den Lernprozess von KNN eingegangen. Ergänzend ist auf [29] zu verweisen.

### 2.4.4 Arten von KNN

Zu den KNN gehören u. a. Perceptrons, Feed Forward Neural Networks, faltende neuronale Netze (engl. Convolutional Neural Networks) (CNN) und Recurrent Neural Networks. Für weitere Informationen hierzu siehe auch [24].

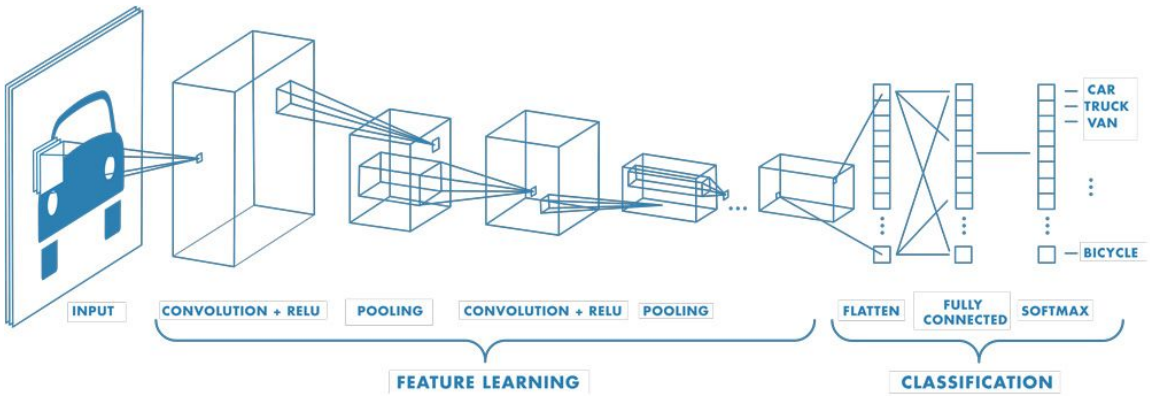


**Abbildung 2.8:** Aufbau eines CNN [30]

Für die Aufgabenbearbeitung im Rahmen dieser Arbeit muss auf die Bildverarbeitung zurückgegriffen werden. Würde für die Lösung dieser Aufgabe auf ein vollvermaschtes künstliches Netzwerk zurückgegriffen werden (siehe Abbildung 2.6), würde das bedeuten, dass das Netz zur Bildverarbeitung eine enorme Anzahl an Neuronen benötigt. Dies ist darin begründet, dass für die Verarbeitung

eines üblichen Full High-Definition (HD) Bildes, mit der Auflösung $1920 \times 1080$ Pixel und 3-RGB-Farbkanälen, 6.220.800 Eingangsneuronen bereitgestellt werden müssten. Aufgrund der Komplexität müssten auch mehr als zwei Zwischenebenen vorhanden sein. Diese Anzahl an Neuronen ist für konventionelle Rechnereinheiten nicht effizient lösbar.

Eine Abhilfe für dieses Problem stellen die faltenden neuronalen Netzwerke (engl. CNN) dar, siehe Abbildung 2.8. Dazu gehört auch ein ResNet-18 Netz, das im Rahmen dieser Arbeit genutzt wird. Diese CNN setzen sich aus verschiedenen Schichten zusammen und sind vom Grundprinzip ein zum Teil lokal vermaschtes neuronales Feedforward-Netz. Die einzelnen Schichten des CNN sind:

- Eingabeschicht (engl. Input Layer),
- Faltungsschicht (engl. Convolutional Layer),
- Pooling-Schicht (engl. Pooling Layer) und
- vollständig vermaschte Schicht (engl. Fully-Connected-Layer).

Die Faltungsschicht und Pooling-Schicht bilden eine Kombination, wobei die Pooling-Schicht immer auf die Faltungsschicht folgt. Diese Kombination kann mehrfach hintereinander vorhanden sein. Da es um sowohl bei der Pooling-Schicht als auch der Faltungsschicht um lokal vermaschte Teilnetze handelt, bleibt die Anzahl an Verbindungen in diesen Schichten selbst bei großen Eingabemengen begrenzt und in einem für konventionelle Rechnersysteme beherrschbaren Rahmen. Den Abschluss des Netzes bildet eine vollständig vermaschte Schicht (engl. Fully-Connected-Layer). Nachfolgend wird auf die Aufgaben der einzelnen Schichten im Kontext eines CNN eingegangen:

Die Eingabeschicht belässt das z. B. zu verarbeitende Bild in der Auflösung $1920 \times 1080$ Pixel mit 3-RGB-Farbkanälen. Auf dieses Bild wird ein Filter angewendet, die erste Faltung beginnt in der Faltungsschicht.

Die Faltungsschicht ist die eigentliche Faltungsebene. Diese ist in der Lage mittels Filter (engl. kernel) in den Eingabedaten einzelne Merkmale zu erkennen und zu extrahieren. Die Filter sind symmetrische, zweidimensionale Matrizen, definierte Größe (Breite x Höhe x Kanäle). Dabei hängen die Werte der Matrizen von den zu filternden oder hervorzuhebenden Strukturen ab. Die Anzahl und Größe der Filter werden je nach Anwendungsfall definiert. Der Filter wird beginnend von oben links über das Bild mit der gewählten Schrittlänge (Schiebeabstand, engl. stride) „geschoben“. Mittels Faltung (Berechnung des Skalarproduktes) wird zwischen dem Bild, genauer gesagt, den Pixel-Werten des Bildes und dem Filter, eine neue Ergebnismatrix gebildet, die auch Feature-Map genannt wird. Diese Feature-Map kann somit Merkmale wie Linien, Kanten oder bestimmte Formen beinhalten. An dieser Stelle sei erwähnt, dass die Werte des Filters auch negativ sein können. Dadurch könnten in der Ergebnismatrix auch negative Pixel-Werte vorkommen, die nicht mehr interpretierbar sind. Daher kommen Aktivierungsfunktionen, wie die Rectified Linear Unit (ReLU) zum Einsatz, wodurch negative Werte mit einer 0 überschrieben werden und positive Pixel-Werte ihren Wert behalten.

Die Pooling-Schicht, auch Subsampling-Schicht genannt, verdichtet und reduziert die Auflösung der erkannten Merkmale, also der Feature-Map. Hierfür verwendet die Schicht verschiedene Methoden, wie das Maximal-Pooling oder das Mittelwert-Pooling. Das Pooling verwirft überflüssige Informationen und reduziert die Datenmenge. Die Leistungsfähigkeit beim ML wird dadurch nicht verringert. Durch die reduzierte Datenmenge erhöht sich die Berechnungsgeschwindigkeit.

Die vollständig vermaschte Schicht schließt sich den sich wiederholenden Abfolgen von Faltungs- und Pooling-Schichten an und kann so der Merkmalsextraktion dienen. Dabei sind alle Merkmale und Elemente der vorgelagerten Schichten mit jedem Ausgabemerkmal verknüpft. Die

vollständig verbundenen Neuronen können in mehreren Ebenen angeordnet sein. Die Anzahl der Neuronen ist abhängig von den Klassen oder Objekten, die das neuronale Netz unterscheiden soll.

Der Vorteil von CNN gegenüber „klassischen“ KNN besteht darin, dass diese sich ausgezeichnet für die Verarbeitung großer Datenmengen eignen. Durch die Faltungsschichten reduziert sich die Speicheranforderung drastisch. Des Weiteren kann auch die Trainingszeit, vordergründig unter Nutzung des Transfer-Lernens, signifikant reduziert werden.

Weitere Erklärungen zu den Pooling-Verfahren, Aktivierungsfunktionen, Kostenfunktionen, Fehlerrückführung (engl. backpropagation), Konvergenz, Optimierungsalgorithmen oder Optimierung der Hyperparameter finden sich in der Literatur [21]. Die Architektur des verwendeten ResNet-18 Netzes wird in dem Kapitel Training des Modells vorgestellt. Die Anwendung der Netze erfolgt u. a. auf dem JetRacer, der im folgenden Kapitel vorgestellt wird.

# 3 JetRacer

Im Rahmen dieser Masterarbeit soll eine KI-gestützte „Follow-Me“ Funktion implementiert werden. Als Beispiel wird hierzu ein JetRacer verwendet, auf dem die Funktionalität implementiert wird. In diesem Kapitel sollen daher der Aufbau, die Hard- und Software sowie mögliche Besonderheiten des JetRacers beschrieben werden. Bevor jedoch tiefer in die Thematik eingestiegen wird, soll in dem nächsten Abschnitt erläutert werden, was genau ein JetRacer ist und wieso er sich für diese Arbeit eignet.

## 3.1 Aufbau des JetRacer – Hardware und Software

Der JetRacer ist im Grunde genommen eine Zusammensetzung von einem Nvidia Jetson Nano, einem Nvidia Jetson Nano Carrier Board, einem RC-Autobausatz, einem JetRacer Expansion Board, einer WLAN-Karte sowie einer Kameraeinheit. Nachfolgend werden die Bestandteile kurz erläutert.


**Abbildung 3.1:** Nvidia Jetson Nano [31]

Der Jetson Nano (siehe Abbildung 3.1) ist ein kleiner, vordergründig grafisch leistungsstarker Computer, mit dem verschiedene neuronale Netze für Anwendungen wie Bildklassifizierung, Objekterkennung, Segmentierung und Sprachverarbeitung parallel ausgeführt werden können. All dies lässt sich in einer einfach zu bedienenden Plattform via Python bedienen. Die Leistungsaufnahme von ca. 5 Watt ermöglicht auch mobile, nicht kabelgebundene Anwendungen, wie innerhalb eines RC-Autos.

Um den Jetson Nano noch mit weiteren Peripheriegeräten verbinden zu können, wird das Jetson Nano Carrier Board, siehe Abbildung 3.2 benötigt. Es bietet eine Vielzahl von Anschlüssen. Unter anderem besitzt das Board einen Steckplatz für den Jetson Nano, eine Netzversorgung für den Jetson Nano, einen Steckplatz für eine WLAN-Karte, einen Ethernet-Anschluss, vier USB, einen HDMI, ein DisplayPort Anschluss, zwei CSI-Kamera Anschlüsse, sowie eine 40-Pin-I/O-Anschlussleiste und weitere mehr. Durch die USB-Anschlüsse und den HDMI-Port ist es möglich, sich direkt mit dem Jetson Nano zu verbinden und darauf zu programmieren.

Für eine fahrende, mobile Anwendung der bereits erläuterten Komponenten ist außerdem ein JetRacer Expansion Board (siehe Abbildung 3.3) notwendig, das u. a. Motor- und Servo-Treiber sowie eine Batterieaufnahme mitbringt, wie er z. B. von Waveshare angeboten wird.

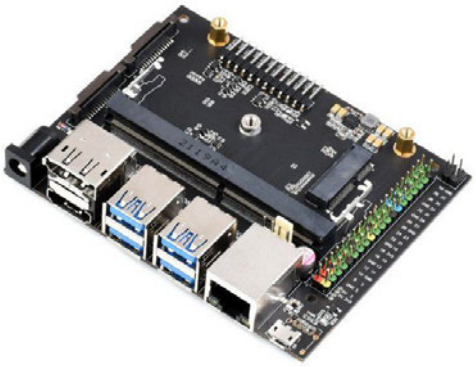

**Abbildung 3.2:** Nvidia Jetson Carrier Board [31]

Der RC-Bausatz, siehe Abbildung 3.4, wird ebenfalls von Waveshare angeboten. Er liefert neben einem Aluminium-Chassis, zwei Antriebsmotoren und einem Servomotor inkl. Lenkgestänge auch eine WLAN-Karte und Reifen mit. Für die Umfelderfassung wird eine 8MP 160°Field of View Kamera genutzt, die ebenfalls in dem RC-Bausatz von Waveshare vorhanden ist.

Auf eine Zusammenbauanleitung aller Komponenten wird im Rahmen dieser Arbeit verzichtet und auf [32] verwiesen. Der zusammengebaute JetRacer ist anhand der Abbildung 3.4 abgebildet.

Um eine Nutzung des JetRacer zu ermöglichen, ist neben der Hardwareinstallation auch eine Softwareinstallation erforderlich. Hierbei gibt es verschiedene Möglichkeiten bzw. Absprungbasen, die nachfolgend kurz erläutert werden sollen. Grundlage des JetRacers bildet ein Linux Ubuntu 18.04 Betriebssystem, das jedoch für den Nvidia Jetson angepasst wurde, wie Nvidia-Treiber, Bootloader, notwendige Firmware, aber auch vorinstallierte Bibliotheken wie TensorRT oder CUDA.

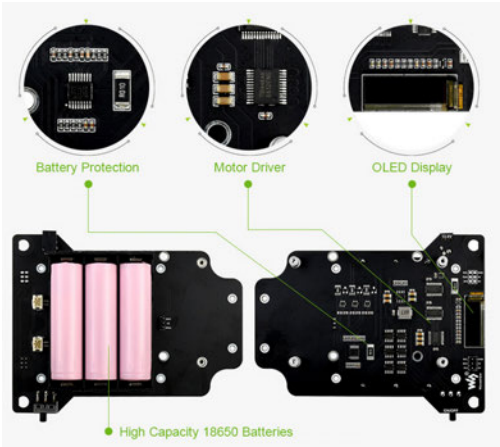


**Abbildung 3.3:** Waveshare Expansion Board [33]

1. Waveshare – Installation mittels Image
   - Quelle:
     `https://www.waveshare.com/wiki/JetRacer_AI_Kit#2._Software_setup`
   - Jetpack Version: 4.5


**Abbildung 3.4:** Waveshare JetRacer RC-Bausatz [33]

- Installierte Bibliotheken: Jetpack, TensorRT, cuDNN, torch2trt, PyTorch, TensorFlow, CUDA, JupyterLab Server, JetCam, OLED-Display Anzeigen, JetRacer inkl. Beispiele sowie auch eine große Anzahl vortrainierter Modelle
- Anmerkungen: Nach der Installation müssen folgende Befehle ausgeführt werden, da sonst die Motortreiber des JetRacers von Waveshare nicht angesteuert werden:
  *cd jetracer; git checkout master; sudo python3 setup.py install; sudo reboot*
- Größe: 13,1 GB
- Aufwand: gering
- Vorteil: Angepasst an den Waveshare JetRacer, keine Komplikationen bei der Installation und Anwendung
- Nachteil: Veraltetet Bibliotheksversionen werden verwendet, z. B. Jetpack Version 4.5 statt der aktuellen 4.6.2

2. JetCard – Installation mittels Image und nach Installation von Bibliotheken
   - Quelle: `https://github.com/NVIDIA-AI-IOT/jetcard`
   - Jetpack Version: 4.5.1
   - Installierte Bibliotheken: PyTorch, TensorFlow, torch2trt, JupyterLab Server, JetCam, OLED-Display Anzeigen, JetRacer sowie die Pretrained-TensorFlow Modelle des Paketes Slim, siehe dazu [34]
   - Anmerkung: Nach der Installation des vorgefertigten Images kommt es zu einem Fehler innerhalb der Torch-Bibliothek, der nicht behoben werden konnte (*aarch64: libgomp.so.1: cannot allocate memory in static TLS block*). Der Fehler konnte reproduziert werden.
   - Größe: 7,6 GB
   - Aufwand: mittel
   - Vorteil: prinzipiell keine Nachinstallation erforderlich
   - Nachteil: in der aktuellen Version JetCard 4.5.1 nicht voll funktionsfähig
3. JetPack – Installation mittels Basis JetPack Image und Nachinstallation aller notwendigen Bibliotheken

- Quelle: `https://developer.nvidia.com/embedded/jetpack-sdk-461`
- Jetpack Version: 4.6.2
- Installierte Bibliotheken: Jetpack, TensorRT, cuDNN, CUDA
- Anmerkung: Nachinstallation der Bibliotheken PyTorch, TensorFlow, torch2trt, JupyterLab Server, JetCam, OLED-Display Anzeigen und JetRacer gestaltet sich mitunter aufwendig, da immer wieder Fehler bei Abhängigkeiten auftreten, die manuell behoben werden müssen.
- Größe: 6,1 GB
- Aufwand: groß
- Vorteil: Verwendung der aktuellsten Bibliotheken
- Nachteil: Aufwendige Nachinstallation erforderlich

Am einfachsten gelingt die Installation über die Nutzung eines vorgefertigten Images über die Variante 1. Das Image ist u. a. bereits auf die Motortreiber des Waveshare-RC-Bausatzes angepasst. Dabei wird das Image mittels einer Software, wie Etcher, auf eine microSD-Karte geflasht. Die microSD-Karte kann anschließend in den SD-Karten Slot des Jetson Nano gesteckt werden.

Nach Anschluss eines Monitors mittels HDMI oder Display-Port-Kabel sowie einer Maus und Tastatur kann des Nvidia Jetson Carrier Board mit Strom versorgt und der Jetson gebootet werden. Im ersten Boot-Vorgang werden weitere Software-Installationsschritte auf dem Monitor angezeigt. Nach Verbindung des JetRacers mit einem Netzwerk, vorzugsweise mittels WLAN, kann über die IP-Adresse des JetRacers (Linux Befehl innerhalb der Konsole *ifconfig*) oder über die Ausgabe auf dem OLED-Display und den Port 8888, sowie über einen beliebigen Browser auf den JetRacer und die Entwicklungsumgebung JupyterLab zugegriffen werden (z. B. `http://<jetson_ip_address>:8888`). Zur Installation des Waveshare Images gibt es auch eine ausführliche Softwareinstallationsanleitung, auf die an dieser Stelle verwiesen werden soll [35].

Für die Masterarbeit wurde neben dem bereits geschilderten ersten Weg der Installation auch die Variante 3. genutzt, siehe Unterabschnitt 4.3.4, um zu prüfen, ob und welche Unterschiede in der Performance zwischen den verschiedenen JetPack-Versionen und Bibliotheksversionen auftreten.

## 3.2 Auswahl des Jetson Nano als Recheneinheit

Bevor im weiteren Verlauf der Arbeit tiefergehend auf die Verwendung des JetRacers eingegangen werden kann, soll zunächst die Frage beantwortet werden, warum die Wahl auf eine Kombination von JetRacer und Jetson Nano als Recheneinheit gefallen ist.

Durch die Anforderungen eines geringen Energieverbrauchs, um für mobile Anwendungen genutzt werden zu können, aber auch hinsichtlich der geringen Kosten sind nur Einplatinenrechner infrage gekommen. Als bekanntester Vertreter für Einplatinenrechner ist der Raspberry Pi zu nennen, der grundsätzlich die Anforderungen bezüglich Energieeffizienz und Kosten erfüllt. Einen wesentlichen Vorteil bietet jedoch der Jetson Nano gegenüber dem Raspberry Pi in Hinblick auf die Anzahl der verfügbaren Grafikeinheiten (engl. Graphics Processing Unit) (GPU). Der Raspberry Pi 4 Mod. B hat als GPU-Chip einen Broadcom VideoCore VI verbaut, der vier Kerne besitzt. [36] Der Jetson Nano hingegen besitzt 128 Kerne, die die Programmierschnittstelle Compute Unified Device Architecture (CUDA) unterstützen. [31] Damit stehen dem Jetson Nano 32-mal mehr GPUs zur Verfügung als dem Raspberry Pi 4 Mod. B, was für den hier angedachten Einsatz einen wesentlichen Vorteil darstellt.

Abstrakt gesehen ist es egal, ob die Berechnungen auf einer Zentraleinheit (engl. Central Processing Unit) (CPU) ausgeführt werden oder auf einer GPU. Das Rechenwerk einer CPU kann die entsprechenden Berechnungen ebenfalls ausführen. Aus Geschwindigkeitsgründen ist es aber sinnvoller, die Berechnungen auf den GPUs auszuführen, da die GPU auf Vektor- und Matrixberechnungen optimiert und die Architektur der GPU auf parallele Verarbeitung ausgelegt ist. Beides sind Eigenschaften, die für KI-Algorithmen, insbesondere für neuronale Netze von enormer Bedeutung sind, da diese im Wesentlichen auf Matrixberechnungen basieren. [37]

Angesichts dessen ist die Wahl auf den Nvidia Jetson Nano gefallen. Für tiefergehende Erläuterungen zu den in KNN zugrunde liegenden Berechnungen und Algorithmen wird auf [37] verwiesen.

**TECHNICAL SPECIFICATIONS**

| | |
|---|---|
| **GPU** | NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores |
| **CPU** | Quad-core ARM Cortex-A57 MPCore processor |
| **Memory** | 4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s |
| **Storage** | 16 GB eMMC 5.1 |
| **Video Encode** | 250MP/sec<br>1x 4K @ 30 (HEVC)<br>2x 1080p @ 60 (HEVC)<br>4x 1080p @ 30 (HEVC)<br>4x 720p @ 60 (HEVC)<br>9x 720p @ 30 (HEVC) |
| **Video Decode** | 500MP/sec<br>1x 4K @ 60 (HEVC)<br>2x 4K @ 30 (HEVC)<br>4x 1080p @ 60 (HEVC)<br>8x 1080p @ 30 (HEVC)<br>9x 720p @ 60 (HEVC) |
| **Camera** | 12 lanes (3x4 or 4x2) MIPI CSI-2 D-PHY 1.1 (1.5 Gb/s per pair) |
| **Connectivity** | Gigabit Ethernet, M.2 Key E |
| **Display** | HDMI 2.0 and eDP 1.4 |
| **USB** | 4x USB 3.0, USB 2.0 Micro-B |
| **Others** | GPIO, $I^2C$, $I^2S$, SPI, UART |
| **Mechanical** | 69.6 mm x 45 mm<br>260-pin edge connector |

**Abbildung 3.5:** Technische Daten des Jetson Nano 4 GB [31]

## 3.3 Notwendige Bibliotheken

Wie bereits in dem Abschnitt der Softwareinstallation genannt, sind für Implementierung und Ausführung der „Follow-Me“-Funktion verschiedene Bibliotheken erforderlich, die in diesem Kapitel vorgestellt werden sollen.

Grundlage für alle Anwendungen stellt das JetPack-Paket dar, das essenziell zum Betreiben des JetRacers ist, da innerhalb dieses Pakets Treiber, Bootloader und Betriebssystem, sowie Basis-Bibliotheken enthalten sind. Da auf die Basis-Bibliotheken nicht weiter eingegangen wird, soll an dieser Stelle auf die JetPack Dokumentation [38] verwiesen werden.

- TensorRT – NVIDIA TensorRT, ein Software Development Kit (SDK) für hochleistungsfähige Deep-Learning-Inferenz, enthält einen Deep-Learning-Inferenzoptimierer und eine Laufzeit, die eine geringe Latenz und einen hohen Durchsatz für Inferenzanwendungen bietet.

- PyTorch – Ist ein Python-Paket, das zwei allgemeine Funktionen bietet: Tensor Berechnung (wie NumPy) mit starker GPU-Beschleunigung und Deep Neural Networks, die auf einem automatischen Differenzierungssystem aufgebaut sind. Es kann genutzt werden, um z. B. vortrainierte Netzwerke mittels Transfer-Lernen für die jeweilige Aufgabe zu trainieren. Transfer-Lernen ist dabei eine Technik zum erneuten Trainieren eines DNN-Modells auf einem neuen Datensatz, was weniger Zeit in Anspruch nimmt als das Trainieren eines Netzwerks von Grund auf. Beim Transfer-Lernen werden die Gewichte eines zuvor trainierten Modells fein abgestimmt, um einen angepassten Datensatz zu klassifizieren. [39]
- Torch – Ist eine Tensor-Bibliothek wie NumPy mit starker GPU-Unterstützung, dabei beinhaltet das Torch-Paket Datenstrukturen für mehrdimensionale Tensoren und definiert mathematische Operationen über diese Tensoren. Ferner bietet es viele Programme für die effiziente Serialisierung von Tensoren (Abbildung von strukturierten Daten auf eine sequenzielle Darstellungsform). Es hat ein CUDA-Gegenstück, mit dem die Tensor Berechnungen auf einer NVIDIA-GPU mit höherer Rechenleistung ausgeführt werden können.
- Torchvision – Das Torchvision-Paket besteht aus gängigen Datensätzen, Modellarchitekturen und gängigen Bildtransformationen für Computer Vision Anwendungen.
- Torch2trt – Ist ein PyTorch-zu-TensorRT-Konverter, der die TensorRT-Python-API verwendet. Die Konvertierung bietet den Vorteil, dass die TensorRT-Modelle speziell für die NVIDIA-Umgebung angepasst sind und dementsprechend mehr Performance liefern.
- JetCam – Stellt eine einfach zu bedienende Python-Kameraschnittstelle für den NVIDIA Jetson dar, sie unterstützt CSI-Kamera-Module, aber auch USB-Kameras.
- JetRacer – Bildet die Schnittstellen für die Längsdynamik und Querdynamik-Regelung des JetRacers zur Verfügung.

## 3.4 Hinweise zur IDE und Nutzung des JetRacer

Als Integrated Development Environment (IDE) wird im Rahmen dieser Arbeit JupyterLab, auf Grundlage von Python genutzt, siehe Abbildung 3.6. Der Aufruf nach Installation des Jupyter-Servers erfolgt über die IP-Adresse des JetRacers und den standardmäßigen Port 8888. z. B. `http://192.168.68.20:8888`. Dabei muss sich das Endgerät, über welches auf den JetRacer zugegriffen werden soll, im selben Netzwerk befinden.

Es besteht auch die Möglichkeit mittels Secure Shell (SSH) eine Verbindung zu dem JetRacer aufzubauen, sofern SSH nicht deaktiviert wurde. Somit können Befehle aus der Ferne ausgeführt werden, ohne dass dabei direkte Eingabe- und Ausgabegeräte an dem JetRacer angeschlossen sind. Die Steuerung erfolgt über die Konsole, z. B. über das Programm Putty.

Um Speicherplatz auf dem JetRacer zu sparen, kann die Desktop-grafische Benutzeroberfläche (engl. Graphical User Interface) (GUI) deaktiviert werden. Um dies zu erreichen, können die Befehle aus Quelltext 3.1 genutzt werden:

```
$ sudo init 3 # temporaere Deaktivierung der Desktop GUI
$ sudo init 5 # Aktivierung der Desktop GUI
$ sudo systemctl set-default multi-user.target # dauerhafte Deaktivierung der
    Desktop GUI
$ sudo systemctl set-default graphical.target  # dauerhafte Aktivierung der
   Desktop GUI
```

**Quelltext 3.1:** Konsolenbefehle zur Aktivierung/Deaktivierung der Benutzeroberfläche

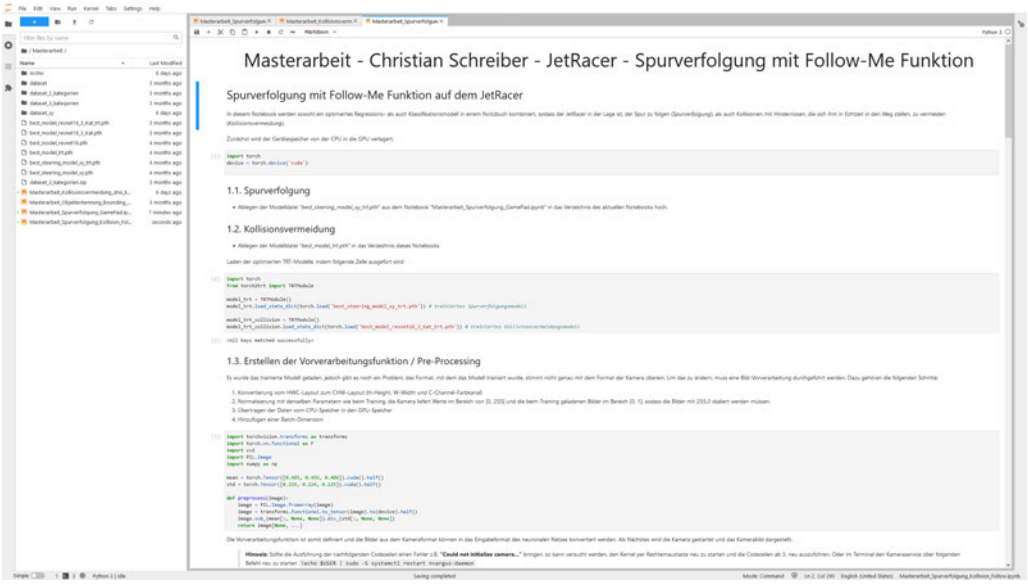


**Abbildung 3.6:** Darstellung der IDE JupyterLab

Sollte es im Fall der Nutzung des JetRacers u. a. beim Zugriff auf die Kamera zu Fehlern kommen, kann dieser Fehler evtl. durch einen Neustart des Kamera-Services behoben werden. Aus der JupyterLab IDE kann der Kameradienst mittels der Codezeile aus Quelltext 3.2 neu gestartet werden, dabei ist nur die Eingabe des Passwortes erforderlich.

```
1 !echo $USER | sudo -S systemctl restart nvargus-daemon
```

**Quelltext 3.2:** Befehl zum Neustart des Kameradienstes innerhalb von JupyterLab

## 3.5 Probleme mit dem JetRacer

Hardwareprobleme, die bei der Nutzung des JetRacers auftraten, lassen sich vorrangig auf eine schlecht justierte bzw. kalibrierte Lenkung zurückführen. Dies äußerte sich durch fehlendes oder unterschiedliches Lenkverhalten bei positiven oder negativen Lenkwinkeln. Zur Abstellung des Fehlers ist darauf zu achten, dass bei Nulllage des Servos beide Vorderräder parallel ausgerichtet sind und keinen Lenkwinkel ausweisen. Im Anschluss ist die Anschlussplatte, siehe Teil 1 in Abbildung 3.7, zur Verbindung des Servomotors mit dem Lenkgestänge, wie anhand der Abbildung 3.7 dargestellt auszurichten.

Weiterhin sollte darauf geachtet werden, dass die Räder nicht zu viel Spiel aufweisen, da sich dies ebenfalls negativ auf die Lenkung auswirken kann.

Ein weiteres Problem, das hauptsächlich bei längerem Betrieb aufgetreten ist, ist eine starke Latenz zwischen der Sensoraufnahme und der Ausgabe, was sich z. B. in einer verzögerten Darstellung des Kamerabildes in JupyterLab äußert. Mögliche Maßnahmen zur Problembehebung sind u. a. die Deaktivierung der Desktop GUI bzw. ein Reboot des Systems.

Des Weiteren kam es zu einem schweren Softwarefehler. Nach Update eines neuen Systems und anschließendem Reboot erfolgte weder ein Starten des Systems, noch wurde eine Fehlermeldung angezeigt. Offensichtlich kam es zu einem Fehler, der zu einem Überschreiben, des Bootloaders geführt hat. Dieser Fehler konnte auch durch ein Einsetzen einer neuen microSD-Karte nicht behoben werden. Die Lösung stellt der Nvidia SDK-Manager dar, der jedoch ausschließlich auf einem anderen Host-System mit einem Ubuntu Version 18.04 ausgeführt werden kann. Die Verbindung mit

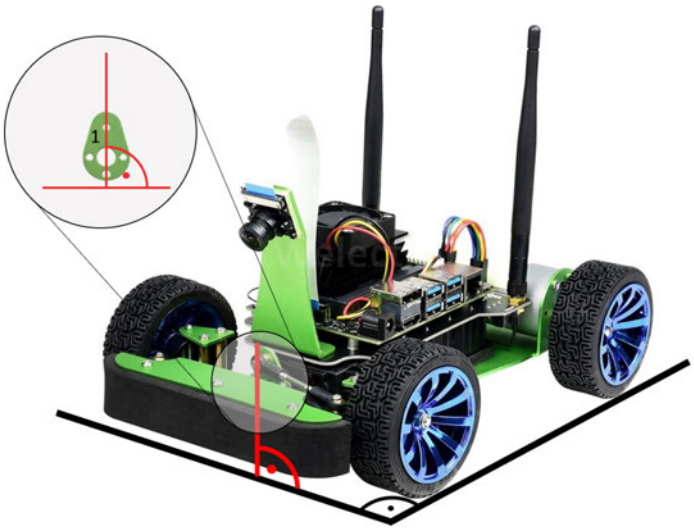

**Abbildung 3.7:** JetRacer Ausrichtung des Lenkgestänges

dem JetRacer wird anschließend mit einem Micro-USB-Kabel hergestellt. Außerdem muss für den Recovery Mode ein Jumper auf dem Nvidia Carrier Board gesetzt werden. Dazu sind die Pins „FC REC“ und „GND“ kurzzuschließen. Im Anschluss kann die Stromversorgung an dem Nvidia Carrier Board hergestellt und nach ca. 30s eine Verbindung zwischen Hostrechner und JetRacer aufgebaut werden. Der SDK-Manger sollte den Jetson Nano Board erkannt haben, den Anweisungen in dem Programm ist Folge zu leisten, um einen erfolgreichen Reset des Jetson Nano zu erreichen. Sollte die Verbindung mit dem SDK-Manger nicht hergestellt werden können, ist die Stromversorgung des Nvidia Carrier Board zu prüfen. Des Weiteren ist es möglich, dass es beim Flashen mit dem SDK-Manager zu Abbrüchen kommt. In diesen Fällen ist der Vorgang zu wiederholen. Eine bebilderte Vorgehensweise ist hier zu finden: [40].

# 4 Implementierung der Follow-Me-Funktion

In den vorangegangenen Kapiteln sind sowohl die verschiedenen Fahrerassistenzsysteme, wie der Spurhalteassistent oder der Abstandsregeltempomat, vorgestellt worden, als auch die Funktionsweise und der Aufbau von KNN. In diesem Kapitel soll die Implementierung der „Follow-Me“-Funktion am Beispiel des JetRacers erläutert werden. Dazu wird, vergleichbar mit den Fahrerassistenzsystemen, die „Follow-Me“-Funktion in die Aufgabe der Spurverfolgung und die Funktion der Abstandsregelung bzw. Hinderniserkennung geteilt. In dem nachfolgenden Abschnitt soll also zunächst erläutert werden, wie die Funktionalität der Spurverfolgung auf dem JetRacer umgesetzt wird. Anschließend steht die Funktion der Hinderniserkennung im Mittelpunkt der Betrachtung, bevor in einem vorletzten Abschnitt eine Fusion beider Funktionen thematisiert werden soll. Den Abschluss bildet eine Betrachtung der Anwendung der JupyterLab Notebooks auf dem JetRacer. Die in den folgenden Abschnitten dargestellten Codeblöcke dienen ausschließlich zur Darstellung und haben keinen funktionalen Anspruch. Der komplette und funktionierend Code ist im Anhang aufgeführt.

## 4.1 Spurverfolgung

Grundlegendes Ziel ist es, dass der JetRacer einer Spur, Fahrbahn oder einer Markierung folgen kann, ohne dabei die Fahrspur oder Markierung zu verlassen. Die Umsetzung soll mittels eines KNN erfolgen, ohne dabei eine separate Bildverarbeitung inkl. Feature-Detektion zu implementieren. Diese Aufgaben sind mittels des KNN abzuwickeln und gehören nach Abbildung 2.5 in den Bereich des überwachten Lernens (Supervised Learning). Im Detail können diese dem Bereich des Deep Learning zugeordnet werden, da sich für diese Aufgabe der Bildverarbeitung die CNN eignen. Die Umsetzung der Spurverfolgung mittels CNN kann dabei wieder in Teilaufgaben, wie Datenerfassung, Training des Modells und Anwendung des Modells unterteilt werden. In den nachfolgenden Unterkapiteln sollen die einzelnen Teilaufgaben näher beschrieben werden.

### 4.1.1 Datenerfassung

Für das Training bzw. Lernen eines CNN sind Daten für die hier gewählte Anwendung in Form von Bildern erforderlich. Die Bilder müssen dabei bereits so gekennzeichnet sein, dass ersichtlich ist, welcher Spur gefolgt werden soll. Dazu muss der JetRacer in verschiedenen Positionen auf der Spur platziert werden, in diesem speziellen Fall markiert eine schwarze Linie die Spur. Innerhalb der Live-Darstellung der Kamera muss der Zielpfad per Maus oder Gamepad ausgewählt werden. Der grüne Punkt im Bild, siehe Abbildung 4.1, symbolisiert dabei die Zielrichtung, in die der JetRacer fahren soll. Jedes Bild wird dabei mit den XY-Koordinaten des im Livebild gesetzten grünen Punktes abgespeichert. Außerdem kann der Lenkwinkel anhand der Fußpunkt-Koordinaten (roter Punkt) berechnet werden, den der JetRacer einschlagen müsste, um der Spur zu folgen.

Für das Training ist eine Vielzahl von Bildern erforderlich. Die Positionierung des JetRacers auf der Spur zur Datenaufnahme gestaltet sich dabei aufwendig. Aufgrund dieses Sachverhalts wurde zur Positionierung des JetRacers ebenfalls das Gamepad verwendet. Dadurch ist es möglich, den JetRacer am Personal Computer (PC) mittels Live-Kamerabilddarstellung in immer wieder verschiedenen Stellungen auf der Spur mit dem Zielpfad aufzunehmen.

Das Abrufen der Bilder von der Kamera (8MP 160° Sichtfeld (engl. Field of View) (FOV)), erfolgt durch einen IMX219 Sensor mit einer max. Auflösung von $3280 \times 2464$ der CSI-Kamera aus der JetCam Bibliothek.

Dabei wird die Auflösung der Bilder auf eine Größe von $224 \times 224$ Pixel reduziert, da eine höhere Auflösung nicht erforderlich bzw. sinnvoll ist, da das CNN Bilder genau dieser Größe erwartet.

Die Steuerung des JetRacers erfolgt dabei über den rechten Gamepad-Stick, während die Steuerung des Zielpfades zur Kennzeichnung (engl. labeling) mittels des linken Gamepad-Sticks erfolgt. Details dazu sind in Abschnitt 4.4 dargestellt. Die Koordinaten werden dabei von der oberen linken Ecke des Bildes gezählt, siehe Koordinatensystem anhand der Abbildung 4.1. Die Bilder inklusive Zielpfad-Koordinaten, die für das nachfolgende Training von Relevanz sind, werden in einem Ordner mit folgender Namensnomenklatur gespeichert: `xy_xPixel_yPixel_eineindeutigerName.jpg`. Wie anhand der Abbildung 4.1 zusehen, setzt sich der Name wie folgt zusammen:
`xy_056_090_66e195a0-f269-11ec-9344-845cf32689f0.jpg`

Für eine gute Spurführung sind mindestens 300 gekennzeichnete Bilder notwendig, dabei sollte auf unterschiedliche Lichtsituationen, Untergründe aber auch verschiedene Umgebungen geachtet werden. Je differenzierter die aufgenommenen Situationen sind, desto robuster wird die trainierte Spurerkennung, da sie das Umgebungsrauschen filtert.

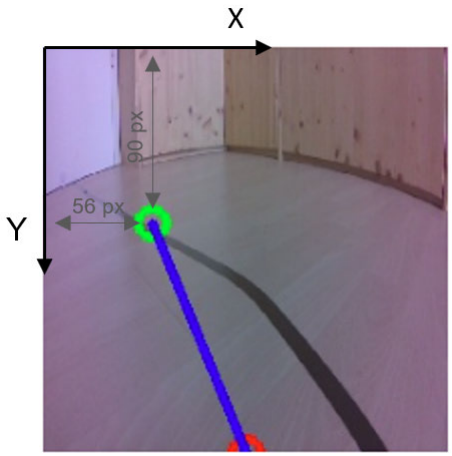


**Abbildung 4.1:** JetRacer Soll-Spurvorgabe mit Koordinatensystem

### 4.1.2 Training des Modells

Das Training des KNN erfolgt anhand der zuvor aufgenommenen Eingangsbilder, die einen Datensatz zusammen mit den XY-Koordinaten aus dem Dateinamen der Bilder bilden. Das Ziel des Trainings ist die Vorhersage der XY-Koordinaten auf Grundlage eines beliebigen Eingangsbildes. Dafür wird das Framework PyTorch genutzt und auf ein ResNet-18 Netz zurückgegriffen.

Bevor das Training beginnen kann, muss noch eine Datenvorverarbeitung implementiert werden. Dabei werden u. a. Funktionen umgesetzt, die für das Laden der Bilder und das Parsen der XY-Koordinaten aus den Dateinamen der Bilder zuständig sind. Es wurde noch eine weitere Option implementiert, sodass die Bilder gespiegelt werden können, um die Eingangsdaten zu vergrößern. Diese Option `random_hflips=true` darf jedoch nur dann auf `true` gesetzt werden, wenn es nicht von Relevanz ist, ob der JetRacer der Spur von links oder rechts folgt. Sollte er eine Fahrbahnmarkierung und dabei nur der rechten Spur folgen sollen, ist diese Option auf `false` zu setzen.

Neben den geschilderten Funktionen sind noch weitere implementiert worden, um die Bilder aufzubereiten, sodass sie von dem PyTorch-Framework verarbeitet werden können. Der dazugehörige Quellcode kann im Anhang eingesehen werden.

Das ResNet-18 Modell erwartet normalisierte Eingangsbilder, in Form von 3-Kanal-RGB-Bildern der Form $(3 \times H \times W)$, wobei „H“ für die Höhe (engl. height) und „W“ für die Breite (engl. width) steht. Dabei soll die Höhe und Breite 224 Pixel betragen. Die Bilder müssen in einem Wertebereich von [0...1] liegen. Demzufolge ist es wichtig, sie mit dem Mittelwert `mean = [0,485, 0,456, 0,406]` und der Standardabweichung `std = [0,229, 0,224, 0,225]` zu normalisieren. Eine Normalisierung hat dabei den weiteren Vorteil, Daten innerhalb eines bestimmten Bereiches zu erhalten, die Schiefe der Bilder zu reduzieren und unnötige Informationen zu entfernen. Dadurch kann das Training schneller abgeschlossen werden. Die Normalisierung reduziert auch die Gefahr von stark abnehmenden oder zunehmenden Gradienten. Die oben dargestellten Werte für den Mittelwert und die Standardabweichung sind die Werte des ImageNet-Datensatzes und können im Fall von, stark zum ImageNet-Datensatz, abweichenden Bildern durch `mean, std= image.mean([1,2]),image.std([1,2])` ersetzt werden.

Anhand der Abbildung 4.2 ist die Architektur des favorisierten KNN ersichtlich. Dieses besitzt, wie in Abbildung 4.2 abgebildet, 16 Faltungsschichten (convolutional layer) und zwei Pooling-Schichten sowie einige vollvermaschte Netzschichten am Ende des Netzes. Die erste Faltungsschicht enthält 64 Filter, die eine Größe von $7 \times 7$ Pixel haben. Es folgt eine Maximal-Pooling Schicht, die die Auflösung der Feature-Map reduziert. Es schließen sich weitere Faltungsschichten an, die entsprechend der Abbildung 4.2 allesamt Filter der Größe $3 \times 3$ Pixel nutzen. Den Abschluss des Netzes bildet eine vollvermaschte Schicht mit 512 Neuronen. Da in dem hier vorliegenden Fall ein Wert für eine XY-Koordinate vorhergesagt werden muss, wird das Modell auf Regression trainiert, sodass das Output-Feature nur zwei Neuronen besitzt.

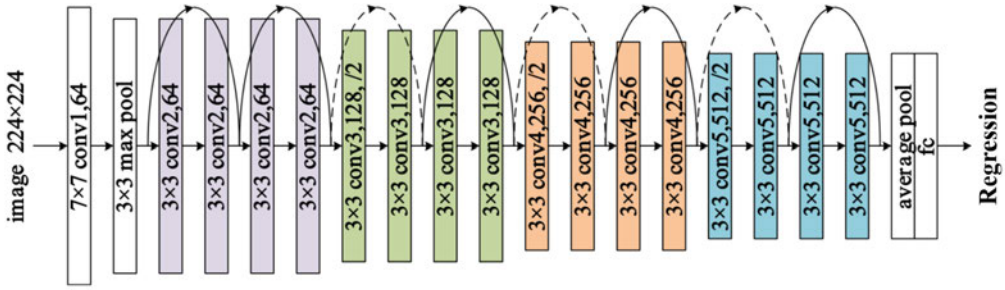


**Abbildung 4.2:** Architektur des ResNet-18 Netzes [41]

Nachdem die Datenvorverarbeitung definiert wurde, muss der Datensatz eingelesen und anschließend aufgeteilt werden, um einen Trainings- und einen Testdatensatz zu generieren. Der Testsatz wird verwendet, um die Genauigkeit des mit dem Trainingsdatensatz trainierten Modells zu überprüfen.

Im Rahmen der Arbeit wurde ein Verhältnis von 90/10 gewählt, 90 % des Datensatzes werden für das Training verwendet und 10 % für das Testen, des Modells.

Der JetRacer besitzt mit seinem Jetson Nano nur einen begrenzten Arbeitsspeicher von 4 GB. Dieser wird sowohl von der CPU als auch der GPU genutzt und ist somit z. B. bereits durch das Betriebssystem reduziert. Angesichts dessen ist es nicht ratsam, die gesamten Trainingsbilder auf einmal in nur einer Trainingsepoche zu verarbeiten. Stattdessen wird zu einer stapelweisen Verbreitung geraten. Dies reduziert die Speicherauslastung und dient der Performance. Dabei wird eine Stapelgröße von acht favorisiert (`batch_size= 8`). Es werden also acht Bilder für den Vorwärts- und

Rückwärtsdurchlauf verwendet. Für das Training von beispielsweise 400 Bildern werden demnach 50 Iterationen benötigt, um eine Epoche zu durchlaufen. Nachfolgende werden die Begriffe Stapel, Iteration und Epoche im Einzelnen beschrieben:

- Eine Epoche ist ein Vorwärts- und Rückwärtsdurchlauf aller Trainingsbeispiele/Bilder.
- Die Stapelgröße ist die Anzahl der Trainingsbeispiele/Bilder, die in einem Vorwärts-/ Rückwärtsdurchlauf verwendet werden. Je höher die Stapelgröße ist, desto mehr Speicherplatz wird benötigt.
- Iterationen stellen die Anzahl der Durchläufe dar, wobei in jedem Durchlauf entsprechend der Stapelgröße die Anzahl von Trainingsbeispielen/Bildern verwendet werden.

Im Anschluss an die angelegten Trainingsdatensätze muss das KNN definiert werden. In dieser Arbeit wird, wie bereits beschrieben, auf ein ResNet-18 Netz zurückgegriffen, das u. a. Bestandteil des PyTorch Frameworks ist. Neben verschiedenen ResNet-Netzen bietet PyTorch auch GoogLeNet-, AlexNet-, MobileNet-Netze und viele mehr an. [39] ResNet-Netze werden für die Verwendung von Objekterkennungsaufgaben empfohlen. [42] Dabei wird im Speziellen auf die Technik des Transfer-Lernen zurückgegriffen. Transfer-Lernen ist eine Methode des Deep Learning und wird insbesondere bei Bild- und Textverarbeitung eingesetzt, da somit bei komplexen tiefen neuronalen Netzen (engl. Deep Neural Network) (DNN) die Trainingszeit signifikant reduziert werden kann. Diese Technik nutzt dabei ein, bereits auf eine Problemstellung trainiertes, Modell. Anschließend wird das Modell beim Transfer-Lernen für die neue Aufgabe genutzt, dabei wird das vortrainierte KNN für die Lösung der neuen Problemstellung angesetzt, siehe Abbildung 4.3. [43]

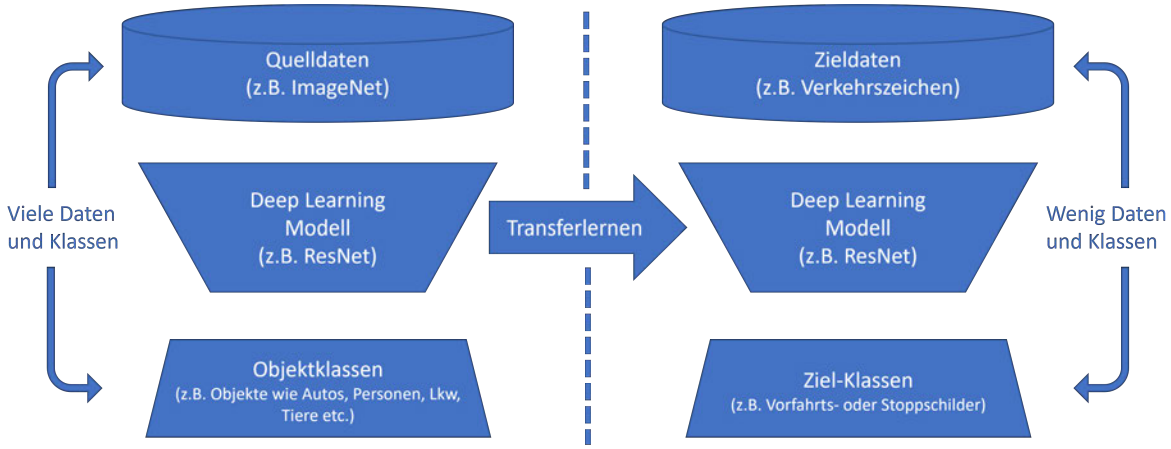


**Abbildung 4.3:** Modell des Transfer-Lernens (engl. transfer learning)

Um diese Methode anzuwenden, wird das Modell mittels des Befehls in Zeile 1 aus Quelltext 4.1 mit der Option `pretrained=true` instanziiert. In dem Beispiel der Spurerkennung wird kein Klassifikator genutzt, sondern die Regression, da anhand des Eingangsbildes Aussagen zu den XY-Koordinaten der zu folgenden Spur gemacht werden sollen.

Aufgrund dessen, dass die letzte Schicht des ResNet-18 Netzes eine vollvernetzte Schicht mit 512 Neuronen ist, um z. B. 512 verschiedene Objekte vorhersagen zu können, wird die letzte Schicht durch eine neue untrainierte Schicht mit nur zwei Neuronen ersetzt. Dadurch können Vorhersagen für zwei Objekte bzw. Werte getroffen werden. Mithilfe des Befehls aus Zeile 2 des Quelltextes 4.1 wird diese vollvernetzte Schicht erstellt.

```
1 model = models.resnet18(pretrained=true)
2 model.fc = torch.nn.Linear(512, 2)
```

```
3 device = torch.device('cuda')
4 model = model.to(device)
```

**Quelltext 4.1:** Auszug aus JupyterLab Notebook - Spurverfolgung - Vorbereitung für Modell-Training

Das Framework PyTorch benutzt das Paket `torch.autograd`, das bei jedem Vorwärtslauf durch das Modell dynamisch einen gerichteten azyklischen Graphen (engl. gerichteten azyklischen Graphen (engl. Directed Acyclic Graph) (DAG) directed acyclic graph) generiert. Dabei handelt es sich um eine besondere Form eines Graphen, basierend auf der Graphentheorie. Diese Form ist dadurch gekennzeichnet, dass die Verbindungen zwischen den Knoten als Kanten bezeichnet werden und eine Richtung besitzen. Schleifen sind bei dieser Form nicht zugelassen. [44] Im Rahmen des Pakets `torch.autograd` werden in dem Vorwärtspfad die Knoten des Graphen als Tensoren und die Kanten als die elementaren Tensor-Operationen definiert. Mittels dieser Informationen können Gradienten aller Tensoren automatisch zur Laufzeit ermittelt und über die Fehlerrückführung (engl. backpropagation) effizient berechnet werden.

Nachdem das Modell generiert wurde, wird es samt kompletten Graphen auf die GPU mit den Befehlen aus Zeile 3 und 4 von Quelltext 4.1 übertragen, um die Vorteile der GPU bei den Matrixberechnungen nutzen zu können.

In direkter Vorbereitung des Trainings werden die Parameter für das Training zur Regression der XY-Koordinaten definiert. Das Training wird auf 50 Epochen angesetzt. PyTorch bietet verschiedene Kosten-/Fehlerfunktionen (engl. loss function) und Optimierungsalgorithmen an. Als Optimierungsalgorithmus wird der Adam-Algorithmus genutzt. Dabei werden die Modellparameter iterativ angepasst, um die Kostenfunktion auf ein Minimum zu reduzieren.

Mit dem Befehl aus Zeile 5 von Quelltext 4.2 wird aus dem PyTorch Framework ein Optimierer instanziiert, der den Adam-Algorithmus verwendet. An dieser Stelle sei erwähnt, dass es auch andere Optimierer gibt, wie den Stochastischen Gradientenabstieg (engl. Stochastic Gradient Descent) (SGD), der im weiteren Verlauf der Arbeit noch erläutert wird. Des Weiteren werden die Modellparameter (die Gewichte der Neuronen) an den Optimierer übertragen.

Das eigentliche Training wird per Iteration über den Trainingsdatensatz mittels einer kopfgesteuerten-Schleife, siehe Zeile 7 in Quelltext 4.2, gestartet. Zu Beginn werden dabei alle Gradienten des Netzwerkgraphen mit dem Befehl aus Zeile 15 von Quelltext 4.2 zurückgesetzt. Die Eingangsbilder und die Kennzeichnungen (Labels) müssen sich für das Training ebenfalls auf der GPU befinden, dies wird über die Zeilen 12 und 13 des Quelltextes 4.2 sichergestellt. Danach werden die Modellparameter (Gewichte) für die Epochen in einem Vorwärtslauf durch den Graphen berechnet. Durch den Befehl aus Zeile 17 des Quelltextes 4.2 werden die Werte für die XY-Koordinaten mit den aktuellen Modellparametern durch das Modell entsprechend den Eingangsbildern des Trainingsdatensatzes vorhergesagt. Nachdem nun die Ergebnisse des Modells mit den aktuellen Parametern vorliegen, muss dieses Ergebnis bewertet werden. Hierzu wird eine Kostenfunktion oder auch Fehlerfunktion (engl. loss function) genutzt. Dabei gibt es verschiedene Funktionen, auf die zurückgegriffen werden kann, siehe [45]. In diesem Teil der Arbeit kommt die Fehlerfunktion des mittleren quadratischen Fehlers (engl. Mean Squared Error) (MSE) zum Einsatz. Aufgrund des quadratischen Terms in der Fehlerberechnung bestraft diese Fehlerfunktion eher große Fehler bzw. Ausreißer als kleine Fehler.

Mit dem Befehl aus Zeile 19 des Quelltextes 4.2 wird der Modellfehler über den MSE berechnet. Die Gradienten werden mit Zeile 22 von Quelltext 4.2 bestimmt und mit dem Befehl in Zeile 24 des Quelltextes 4.2 schließlich die Parameter (Gewichte) des Modells mittels Fehlerrückführung (engl. backpropagation) aktualisiert.

Im Anschluss erfolgt die Validierung des eben trainierten Modells mittels des Testdatensatzes, siehe Quelltext 4.2 Zeile 26 - 33. Dabei werden die gleichen bereits durchgeführten Schritte ausgeführt: Iteration über den Testdatensatz, Übertragung der Bilder inkl. Kennzeichnung (Soll-Spur) auf die GPU, Vorhersage der XY-Koordinaten durch das zuvor trainierte Modell und Berechnung des Testfehlers. Auf die Schritte der Fehlerrückführung und Aktualisierung der Modellparameter wird bei der Validierung bewusst verzichtet.

Zum Abschluss wird die Testgenauigkeit in Bezug auf die aktuellen Modellparameter über die Zeile 34 des Quelltextes 4.2 berechnet. Ist der Testfehler geringer als der bisher beste Fehlerwert aus den vorherigen Epochen, wird das aktuelle Modell gespeichert und die nächste Epoche des Trainings beginnt.

Nach Durchlaufen aller Epochen ist das Training abgeschlossen. Der Durchlauf der 50 Epochen für das Training des Netzes für die Spurverfolgung benötigte auf dem Jetson Nano ca. 30 min.

```
1  NUM_EPOCHS = 50
2  BEST_MODEL_PATH = 'best_steering_model_xy.pth'
3  best_loss = 1e9
4
5  optimizer = optim.Adam(model.parameters())
6
7  for epoch in range(NUM_EPOCHS):
8      model.train()
9      train_loss = 0.0
10     for images, labels in iter(train_loader):
11         # get the inputs
12         images = images.to(device)
13         labels = labels.to(device)
14         # zero the parameter gradients
15         optimizer.zero_grad()
16         # predict classes using images from the training set
17         outputs = model(images)
18         # compute the loss based on model output and real labels
19         loss = F.mse_loss(outputs, labels)
20         train_loss += float(loss)
21         # backpropagate the loss
22         loss.backward()
23         # adjust parameters based on the calculated gradients
24         optimizer.step()
25
26     model.eval()
27     test_loss = 0.0
28     for images, labels in iter(test_loader):
29         images = images.to(device)
30         labels = labels.to(device)
31         outputs = model(images)
32         loss = F.mse_loss(outputs, labels)
33         test_loss += float(loss)
34     test_loss /= len(test_loader)
35
36     print('%f, %f' % (train_loss, test_loss))
```

```
37     if test_loss < best_loss:
38         torch.save(model.state_dict(), BEST_MODEL_PATH)
39         best_loss = test_loss
```

**Quelltext 4.2:** Auszug aus JupyterLab Notebook - Spurverfolgung - Training

### 4.1.3 Anwendung des Modells

Nachdem das Modell trainiert wurde, kann es auf dem JetRacer angewendet werden. Mittels des trainierten Modells und anhand der von der Kamera aufgenommenen Bilder kann eine Vorhersage zu der zu folgenden Spur getroffen werden. Dazu wird erneut ein ResNet-18 Modell mit den zuvor trainierten Gewichten geladen und aus Performancegründen an die GPU übergeben, siehe Zeile 1 - 3 des Quelltextes 4.3.

```
1  model.load_state_dict(torch.load('best_steering_model_xy.pth'))
2  device = torch.device('cuda')
3  model = model.to(device)
4  model = model.eval().half()
5
6  def preprocess(image):
7      image = PIL.Image.fromarray(image)
8      image = transforms.functional.to_tensor(image).to(device).half()
9      image.sub_(mean[:, None, None]).div_(std[:, None, None])
10     return image[None, ...]
```

**Quelltext 4.3:** Auszug aus JupyterLab Notebook - Spurverfolgung - Anwendung

Durch den Befehl aus Zeile 4 des Quelltextes 4.3 wird das Modell in den Evaluierungsmodus (engl. Inference-Mode) geschaltet. Die Option `half()` reduziert die Genauigkeit, was zu einer geringen Verschlechterung der Modellgenauigkeit führt, gleichzeitig aber den Modelldurchsatz, der für die Echtzeitverarbeitung der Bilder relevant ist, erhöht.

Für die Verarbeitung der Bilder der Kamera ist, wie bereits erwähnt, eine Vorverarbeitung erforderlich, da das Format der Kamera nicht mit dem Format, mit dem das Modell trainiert wurde, übereinstimmt. Hierzu wird eine Funktion `preprocess(image)`, siehe Zeile 6 des Quelltextes 4.3 definiert. Diese konvertiert das Bild der Kamera aus dem HWC-Format (Höhe, Breite, Kanal (engl. height, width, channel)) in das CHW-Format (Kanal, Höhe, Breite), das von PyTorch gefordert wird. Auch eine Bildnormalisierung wird erneut angewendet, um bessere Ergebnisse aus dem Modell zu erhalten.

Nachdem die Vorverarbeitungsfunktion implementiert wurde, können Instanzen der Kamera und des JetRacers erstellt werden, wobei erneut auf die Bibliotheken von Nvidia, JetCam und JetRacer zurückgegriffen wird. Somit kann mit einfachen Mitteln auf die Kamera und die Antriebseinheiten (E-Motoren und Servo-Motor) des JetRacers zugegriffen werden.

Für die Visualisierung der Regler zur Steuerung des JetRacers wird erneut die Bibliothek ipywidgets genutzt. Details hierzu werden im Abschnitt 4.4 genannt. Der Lenkungsverstärkungsfaktor, welcher u. a. dargestellt wird, kann je nach JetRacer variieren und ist auf den jeweiligen JetRacer anzupassen, damit das Fahrzeug seinen individuellen max. Lenkausschlag nutzt. Außerdem wird der Lenkwinkel, den das Modell als Ergebnis vorhersagt, dargestellt.

Die Regelung der Geschwindigkeit zeigte beim Testen, dass diese mitunter dynamisch gesteuert werden muss. Zum Beispiel kann die Zielgeschwindigkeit bei kleinen Lenkwinkeln wesentlich höher sein, als bei großen Lenkwinkeln, da aufgrund des eingeschränkten Sichtbereichs der Kamera

bereits die Spur rasch aus dem Fokus verloren werden kann. Im ersten Schritt wurde dazu einfach die Nutzung des Gamepads zur dynamischen Regelung der Geschwindigkeit des JetRacers genutzt, die Lenkung blieb autonom. Die dazu genutzten Codeblöcke werden in Abschnitt 4.4 dargestellt.

Im Detail werden die Ergebnisse aus dem ResNet-18 Netz unter Eingabe des aktuellen Kamerabildes innerhalb der in Zeile 1 des Quelltextes 4.4 dargestellten Funktion definiert. Dabei werden die vorhergesagten XY-Koordinaten des Modells mit dem Befehl aus Zeile 4 des Quelltextes 4.4 abgerufen. Die Ergebnisse des Modells für die Koordinaten sind in einem Array gespeichert. Es handelt sich bei dieser Codezeile um einen essenziellen Codeblock, weshalb die Zusammensetzung nachfolgend detailliert dargestellt werden soll.

Das Modell speichert prinzipiell die Ergebnisse in einem `torch.tensor`. Das entspricht einer Matrix und des dazugehörigen Rechengraphs, der zu dieser Matrix führt. Dieser Rechengraph muss mittels der Funktion `detach()` getrennt werden. Anschließend werden die Ergebnisse im Gleitkomma-Format unter Zuhilfenahme der Funktion `float()` umgewandelt. Das Modell und die Ergebnisse befanden sich bisher im GPU-Speicher und müssen nun mittels der Funktion `.cpu()` auf die CPU und auch explizit in den CPU-Speicherbereich transferiert werden, damit die Funktion `numpy()` genutzt werden kann. Die Funktion `numpy()` ist nicht für das Nvidia CUDA Toolkit implementiert worden. Um die Dimensionalität des Numpy-Arrays zu eliminieren, wird die Funktion `flallten()` angewendet. So kann z. B. auf die x-Koordinaten des Modells über den Befehl aus Zeile 6 des Quelltextes 4.4 zugegriffen werden.

Anschließend wird mittels einer geometrischen Berechnung der Lenkwinkel bestimmt, siehe Quelltext 4.4 Zeile 9. Dieser Lenkwinkel wird entsprechend Zeile 12 des Quelltextes 4.4 dem `steering`-Slider zugewiesen. Zur fortlaufenden Verarbeitung der aktuellen Kamerabilder muss folgender Code in Zeile 15 von Quelltext 4.4 ausgeführt werden. Ohne diese Codezeile wird ein Kamerabild nur einmalig abgerufen, es würde keine Aktualisierung stattfinden. Durch die erwähnte Codezeile wird eine Art Dauerschleife gestartet, die den Codeblock immer wieder ausführt, sobald ein neues Bild von der Kamera eingelesen wird.

```
1  def execute(change):
2
3      image = change['new']
4      xy = model(preprocess(image)).detach().float().cpu().numpy().flatten()
5
6      x = xy[0]
7      y = (0.5 - xy[1]) / 2.0
8
9      angle = np.arctan2(x, y)
10     pid = angle * steering_gain_slider.value + (angle - angle_last) *
       steering_dgain_slider.value
11
12     steering_slider.value = pid + steering_bias_slider.value
13
14 execute({'new': camera.value})
15 camera.observe(execute, names='value')
```

**Quelltext 4.4:** Auszug aus JupyterLab Notebook - Spurverfolgung - Anwendung

### 4.1.4 Ergebnisse der Spurverfolgung

Im Rahmen der Arbeit konnte ein Modell implementiert werden, womit der JetRacer einer Spur erfolgreich folgt. Dabei konnten Grenzen festgestellt werden, die hauptsächlich auf die Kameraaufnahmen zurückzuführen sind. Das System hat vor allem Probleme, wenn neben der zu folgenden Spur eine vermeintliche Spur z. B. durch eine Schattenbildung durch einen Türspalt, siehe Abbildung 4.4, vorhanden ist. Des Weiteren kann ein leichtes Pendeln der Regelung um die Spur festgestellt werden, was mittels der Regler „steering kd“ oder „steering gain“ angepasst werden kann. Jedoch hat dies Einfluss auf die Lenkbarkeit des JetRacers im Allgemeinen, mit geringeren „steering gain“, reduziert sich der max. erreichbare Lenkwinkel.

Kommt es zu einem ungewünschten Lenkverhalten des JetRacers in einer bestimmten Situation/Umfeld, so sind von dieser Situation weitere Aufnahmen inkl. Kennzeichnung der richtigen Spur aufzunehmen und das Modell erneut zu trainieren. Über dieses Vorgehen kann das Regelverhalten sukzessiv verbessert werden. Für die weitere Verwendung als Bestandteil der „Follow-Me“-Funktion kann die implementierte Spurverfolgung genutzt werden. Das nächste Kapitel widmet sich der Funktion der Hinderniserkennung und Abstandsregelung.

Das zu diesem Kapitel gehörende JupyterLab Notebook befindet als PDF-Export im Anhang.

**Abbildung 4.4:** Probleme bei der Spurverfolgung - Fehlinterpretation des Türspalts als Spur

## 4.2 Hinderniserkennung

Nachdem eine Spurverfolgung des JetRacers realisiert werden konnte, soll in einem nächsten Schritt eine Abstandsregelung implementiert werden. Dazu wird eine Hinderniserkennung, die auf einer Objekterkennung aufbaut, als Basis genutzt. Der Gedanke dahinter ist, dass wenn ein bestimmtes eindeutiges Objekt z. B. ein Kfz-Kennzeichen erkannt werden kann, handelt es sich um ein Pkw und entsprechend der Größe des erkannten Kennzeichens kann eine Geschwindigkeits-/Abstandsregelung implementiert werden. Dabei sollte im Fall der Nutzung eines Kfz-Kennzeichens als Referenzobjekt beachtet werden, dass verschiedene Kfz-Kennzeichen genutzt werden, damit das Modell nicht eine bestimmte Buchstaben-Zahlen-Kombination eines Kennzeichens lernt und im Fall eines anderen Kennzeichens, dieses nicht als solches erkennt.

Im Kontext dieser Arbeit wird als Erkennungsmerkmal ein Stoppschild, siehe Abbildung 4.5, genutzt, damit nicht mehrere verschiedene Kfz-Kennzeichen trainiert werden müssen. Das Verhalten der Stoppschild-Erkennung ist dabei analog der Kfz-Kennzeichenerkennung.

Um im weiteren Verlauf der Arbeit für die Implementierung der „Follow-Me“-Funktion eine Regelung der Geschwindigkeit des JetRacers auf Basis des Abstandes zu dem Stoppschild zu realisieren, wird das Stoppschild in drei Kategorien bzw. Abständen aufgenommen. Details dazu werden im folgenden Abschnitt der Datenerfassung für die Hindernis- bzw. Objekterkennung vorgestellt.

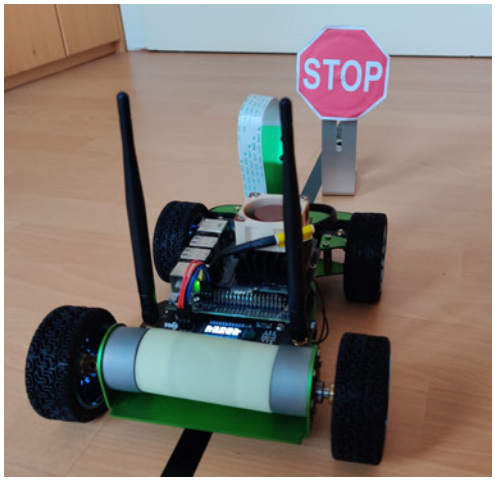


**Abbildung 4.5:** Abbildung des JetRacers mit Stoppschild in blockierter Position

### 4.2.1 Datenerfassung

Für die Bilder inklusive ihrer Kennzeichnung wird wieder das Gamepad genutzt. Wie dieses verwendet werden kann und welche Besonderheiten es gibt, wird in Abschnitt 4.4 dargelegt. Die Bildaufnahme wird analog Unterabschnitt 4.1.1 durchgeführt, lediglich die Kennzeichnung erfolgt anders. Wurde in der Spurerkennung in den Dateinamen des Bildes die Zielspurkoordinaten gespeichert, werden die Bilder hier in drei verschiedene Ordner abgespeichert, die drei Kategorien (frei, langsam und blockiert) darstellen. Dabei bedeuten diese Kategorien:

- `blocked_dir` – blockiert – Für Bilder, die eine Blockade darstellen, der JetRacer muss stehen bleiben
- `slow_dir` – langsam – Für Bilder, die auf eine Reduzierung der Geschwindigkeit des JetRacers hinweisen
- `free_dir` – frei – Für Bilder, die ein freies Umfeld abbilden, der JetRacer kann sich mit seiner max. Geschwindigkeit fortbewegen.

Die Abbildung 4.5 stellt eine Situation dar, in der das Stoppschild so nah ist, dass eine Blockierung vorhanden ist.

Mittels dieser Methodik können die Bilder für die Objekterkennung aufgenommen und gekennzeichnet werden. Dabei ist auf verschiedene Perspektiven in ähnlichen Abständen zu achten, aber auch die Beleuchtung, Hinter- und Untergründe sind zu variieren, um eine robuste Objekterkennung zu realisieren.

Für eine robuste Objekterkennung sollten min. 300 Bilder pro Kategorie aufgenommen werden. Für diese Arbeit entstanden ca. 350 Bilder pro Kategorie. Nachdem die erforderlichen Bilder erfasst wurden, folgt das Training des Models für die Objekterkennung im nächsten Kapitel.

### 4.2.2 Training des Modells

Das Paket `torchvision` beinhaltet eine Vielzahl von Werkzeugen, u. a. zum Erstellen von Datensätzen. Wie bereits in Unterabschnitt 4.1.2 wird auch innerhalb der Objekterkennung der Datensatz in Trainings- und Testdaten aufgeteilt. Auch hier werden die Bilder vor der weiteren Verarbeitung normalisiert, um eine weitere Verarbeitung so effizient wie möglich zu gestalten. Als Mittelwert und Standardabweichung werden erneut die Werte des ImageNet-Datensatzes verwendet. Mittels der Funktion `torch.utils.data.random_split(dataset, [len(dataset) - 90, 10])` wird der Datensatz zufällig in einen 90 % Trainingsdatensatz und 10 % Testdatensatz aufgeteilt.

Analog dem Unterabschnitt 4.1.2 wird auch bei der Objekterkennung eine stapelweise Verarbeitung der Daten realisiert. Die Stapelgröße wird auch in diesem Abschnitt auf acht gesetzt, analog dem Unterabschnitt 4.1.2. Der `DataLoader` ist eine Iterator-Klasse, die einzelne Stapel (engl. batches) des Datensatzes generiert und in den Speicher lädt, sodass große Datensätze nicht vollständig geladen werden müssen. Über die Option `num_workers` kann ausgewählt werden, ob mehrere parallele Ausführungen (engl. threads) gestartet werden sollen, oder ob der Datensatz für jede Epoche neu gemischt werden soll (`suffle=True`).

Mit der Festlegung der Trainings- und Testdatensätze muss das KNN definiert werden. Auch hier eignen sich, für die Objekterkennung, die ResNet-Netze aus dem PyTorch Framework. Die Wahl ist dabei ebenfalls auf ein ResNet-18 Netz gefallen, das bereits vortrainiert ist, um u. a. das Transfer-Lernen zu nutzen. Über den Befehl aus Zeile 1 von Quelltext 4.5 wird das Modell instanziiert.

Aufgrund dessen, dass die letzte Schicht des Netzes eine vollvernetzte Schicht mit 512 Neuronen ist, um z. B. 512 verschiedene Objekte vorherzusagen, wird die letzte Schicht durch eine neue untrainierte Schicht mit nur drei Neuronen ersetzt. Durch diese Maßnahme können Vorhersagen für drei Objekte, in dem speziellen Fall die drei Kategorien, realisiert werden. Mithilfe des Befehls aus Zeile 2 des Quelltextes 4.5 wird diese vollvernetzte Schicht erstellt.

```
1 model = models.resnet18(pretrained=True)
2 model.fc = torch.nn.Linear(512, 3)
3 device = torch.device('cuda')
4 model = model.to(device)
```

**Quelltext 4.5:** Auszug aus JupyterLab Notebook - Hinderniserkennung - Vorbereitung für Modell-Training

Wie bereits in Unterabschnitt 4.1.2 beschrieben, wird auf das Paket `torch.autograd` aus dem Framework PyTorch, erneut zurückgegriffen. Im Rahmen des Paket `torch.autograd` werden in dem Vorwärtspfad die Konten des Graphen als Tensoren und die Kanten als die elementaren Tensor-Operationen definiert. Mittels dieser Informationen können Gradienten aller Tensoren automatisch zur Laufzeit ermittelt und über die Fehlerrückführung (engl. backpropagation) effizient berechnet werden.

Nachdem das Modell generiert wurde, wird es mit den kompletten Graphen, wie bereits in Unterabschnitt 4.1.2 beschrieben, auf die GPU mit den Befehlen aus Zeile 3 und 4 von Quelltext 4.5 übertragen, um die Vorteile der GPU bei der Parallelisierung nutzen zu können. Abschließend werden die (Hyper-)Parameter für das Training zur Klassifikation der drei Kategorien definiert.

Das Training wird auf 30 Epochen angesetzt. Als Optimierungsalgorithmus wird der stochastische Gradientenabstieg mit Impuls (engl. stochastic gradient descent (SGD) with momentum) genutzt. Dabei werden Modellparameter iterativ angepasst, um die Kostenfunktion auf ein Minimum zu reduzieren. Als Gradient wird dabei die Änderung der Gewichtung in Bezug auf die Änderung von

Fehlern bezeichnet. Als Analogie wird dabei gern die Interpretation des Gradienten als Steigung bzw. Gefälle einer Funktion genommen, wobei die Steigung dabei z. B. je nach Größe des Gradienten steiler ausfällt. Im Kontext der KNN ist dies eine günstige Bedingung für das Modell, da es schnell lernt. Wohingegen das Modell aufhört zu lernen, wenn die Steigung bzw. der Gradient null wird.

Ein wichtiger Parameter ist dabei die Lernrate oder auch Schrittweite zwischen den Iterationen. Ist die Lernrate zu hoch, kann über ein Minimum iteriert werden, ist die Lernrate zu klein, kann der Gradientenabstieg in einem lokalen Minimum stecken bleiben. Um sicherzustellen, dass der Gradientenabstieg funktioniert, wird eine Kostenfunktion während der Optimierung genutzt. Dabei werden auf der x-Achse die Wiederholungen abgebildet, während auf der y-Achse die Werte der Kostenfunktion dargestellt werden. Auf diese Weise können die Ergebnisse des Gradientenabstiegs nach jeder Iteration bewertet werden. Dabei konvergiert der Gradientenabstieg, wenn die Kostenfunktion auf einem Niveau stagniert. Im Rahmen dieser Arbeit wird der SGD mit Impuls genutzt. Dabei kann der Impuls als eine Art Trägheit interpretiert werden, der dazu beiträgt, dass der Gradientenabstieg schneller konvergiert und lokale Minima oder Plateaus überspringt, was zu einer schnelleren Konvergenz führt.

Mit dem Befehl aus Zeile 5 von Quelltext 4.6 wird aus dem PyTorch Framework ein Optimierer instanziiert, der den SGD mit Impuls/Momentum verwendet. Die Lernrate wurde mit `lr= 0.001` festgelegt und der `momentum` Parameter mit 0.9. Die Anpassung bzw. Optimierung dieser Parameter, die auch Hyperparameter genannt werden, trägt maßgeblich zu Trainingszeit bei. Des Weiteren werden dem Optimierer die Modellparameter (die Gewichte der Neuronen) übergeben.

```
1  NUM_EPOCHS = 30
2  BEST_MODEL_PATH = 'best_model_resnet18_3_Kat.pth'
3  best_accuracy = 0.0
4
5  optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
6
7  for epoch in range(NUM_EPOCHS):
8
9      for images, labels in iter(train_loader):
10         # get the inputs
11         images = images.to(device)
12         labels = labels.to(device)
13         # zero the parameter gradients
14         optimizer.zero_grad()
15         # predict classes using images from the training set
16         outputs = model(images)
17         # compute the loss based on model output and real labels
18         loss = F.cross_entropy(outputs, labels)
19         # backpropagate the loss
20         loss.backward()
21         # adjust parameters based on the calculated gradients
22         optimizer.step()
23
24     test_error_count = 0.0
25     for images, labels in iter(test_loader):
26         images = images.to(device)
27         labels = labels.to(device)
28         outputs = model(images)
29         test_error_count += float(torch.sum(torch.abs(labels - outputs.argmax
       (1))))
30
```

```
31      test_accuracy = 1.0 - float(test_error_count) / float(len(test_dataset))
32      print('%d: %f' % (epoch, test_accuracy))
33      if test_accuracy > best_accuracy:
34          torch.save(model.state_dict(), BEST_MODEL_PATH)
35          best_accuracy = test_accuracy
```

**Quelltext 4.6:** Hinderniserkennung - Training - Auszug aus JupyterLab Notebook

Das eigentliche Training wird per Iteration über den Trainingsdatensatz mittels einer kopfgesteuerten Schleife, siehe Zeile 7 in Quelltext 4.6, gestartet. Zu Beginn werden dabei alle Gradienten des Netzwerkgraphen, mit dem Befehl aus Zeile 14 von Quelltext 4.6, zurückgesetzt. Die Eingangsbilder und die Kennzeichnungen (Labels) müssen sich für das Training ebenfalls auf der GPU befinden. Dies wird über Zeile 11 und 12 des Quelltextes 4.6 sichergestellt.

Danach werden die Modellparameter (Gewichte) in einem Vorwärtslauf durch den Graphen für die Epochen berechnet. Durch den Befehl aus Zeile 16 des Quelltextes 4.6 werden die Klassen mit den aktuellen Modellparametern durch das Modell entsprechend den Eingangsbildern des Trainingsdatensatzes vorhergesagt.

Nachdem nun die Ergebnisse des Modells mit den aktuellen Parametern vorliegen, muss dieses Ergebnis bewertet werden. Hierzu wird wiederholt eine Fehlerfunktion genutzt. Für diesen Abschnitt der Arbeit kommt die Fehlerfunktion der Kreuzentropie (engl. cross-entropy) zur Anwendung. Diese berechnet einen Bewertungswert (engl. score), der die durchschnittliche Differenz zwischen den vorhergesagten und den tatsächlichen Werten zusammenfasst. Das Ergebnis liegt in dem Bereich zwischen 0 und 1. Wobei 0 eine perfekte Vorhersage symbolisiert. Der Vorteil gegenüber anderen Verlustfunktionen liegt bei der Kreuzentropie darin, dass vor allem die Ergebnisse „bestraft“ werden, die eine hohe Sicherheit aufweisen, aber falsch sind. Gleichzeitig werden aber auch Vorhersagen bestraft, die zwar richtig sind, aber eine geringe Sicherheit des Ergebnisses aufweisen.

Mit dem Befehl aus Zeile 18 in dem Quelltext 4.6 wird der Modellfehler über die Kreuzentropie berechnet. Die Gradienten werden mit Zeile 20 von Quelltext 4.6 bestimmt und mit dem Befehl in Zeile 22 des Quelltextes 4.6 schließlich die Parameter (Gewichte) des Modells mittels Fehlerrückführung aktualisiert.

Im Anschluss erfolgt die Validierung des eben trainierten Modells mittels des Testdatensatzes, siehe Quelltext 4.6 Zeile 24 - 29. Dabei werden die gleichen Schritte wie zuvor durchgeführt: Iteration über den Testdatensatz, Übertragung der Bilder und Kennzeichnung auf die GPU, Vorhersage der Klasse durch das zuvor trainierte Modell und Berechnung des Testfehlers. Auf die Schritte der Fehlerrückführung und Aktualisierung der Modellparameter wird, wie bereits in Unterabschnitt 4.1.2, bei der Validierung bewusst verzichtet.

Zum Abschluss wird die Testgenauigkeit in Bezug auf die aktuellen Modellparameter über die Zeile 31 des Quelltextes 4.6 berechnet. Ist die Testgenauigkeit des aktuellen Modells besser als die Genauigkeit des vorherigen Modells aus der Epoche zuvor, wird das aktuelle Modell gespeichert und die nächste Epoche des Trainings beginnt. Nachdem alle Epochen durchlaufen wurden, ist das Training abgeschlossen.

### 4.2.3 Anwendung des Modells

Die Anwendung des Modells gestaltet sich ähnlich, wie es bereits in Unterabschnitt 4.1.3 umgesetzt wurde. Hierzu muss wie ein ResNet-18 Modell geladen werden. Die Option zur Anwendung des Transfer-Lernens (`pretrained=false`) muss deaktiviert und als letzte Schicht eine vollvermaschte Schicht mit 3 Ausgangsneuronen geladen werden, um eine Vorhersage, der drei Kategorien zu realisieren. Anschließend werden die gelernten Parameter/Gewichte aus dem vorherigen Abschnitt mittels `model.load_state_dict(torch.load('best_model_resnet18_3_Kat.pth'))` geladen und in die GPU übertragen. Das Modell wird in den Evaluierungsmodus mit reduzierter Genauigkeit geschaltet, um den Modelldurchsatz für die Echtzeitverarbeitung zu erhöhen. Wie bereits in Unterabschnitt 4.1.3 beschrieben, folgt wieder eine Definition einer Vorverarbeitungsfunktion für die Bildverarbeitung. Des Weiteren wurde eine kleine GUI implementiert, um eine Visualisierung der aktuellen Vorhersagekategorien zu realisieren. Dazu wird auf drei Regler aus der Bibliothek von ipywidgets zurückgegriffen. Die Steuerung des JetRacers selbst erfolgt dabei mittels des Gamepads. Die Umsetzung dieser Steuerung gleicht dabei der Umsetzung aus dem Unterabschnitt 4.1.1 und wird in Abschnitt 4.4 beschrieben.

Das Modell liefert einen Vektor entsprechend der drei Kategorien mit drei reellen Werten. Dabei geben die reellen Werte noch nicht direkt Aufschluss über die Wahrscheinlichkeit der einzelnen Kategorien. Hierzu wird eine Softmax-Funktion zu Hilfe genommen. Dabei nimmt die Softmax-Funktion die Summe der Werte der Neuronen der Output-Schicht und normalisiert diese auf 1, wodurch eine Wahrscheinlichkeitsverteilung realisiert wird. Der Befehl dazu lautet, `y = F.softmax(y, dim=1)`, wobei das Argument `dim=1` die Achse des Tensors angibt, die normalisiert werden soll. Damit kann eine Wahrscheinlichkeit für das Auftreten jeder Kategorie ausgegeben werden. Über die Funktion `y.flatten()[0]` wird aus dem Tensor ein Array erstellt, auf welches per Indizierung zugegriffen werden kann.

Zur fortlaufenden Verarbeitung der aktuellen Kamerabilder über das Modell wird die folgende Codezeile ausgeführt: `camera.observe(execute, names='value')`. Anschließend kann entsprechend den Wahrscheinlichkeiten für das Auftreten der verschiedenen Kategorien die max. Geschwindigkeit des JetRacers gesetzt werden. Ist zum Beispiel die Wahrscheinlichkeit, dass eine Blockade erkannt wurde, größer als 0.6, wird die max. Geschwindigkeit des JetRacers auf 0 gesetzt, wodurch der JetRacer stoppt (`jetracer.throttle_gain = 0.0`).

### 4.2.4 Ergebnisse der Objekterkennung

Mithilfe des zuvor beschriebenen Modells konnte eine Objekterkennung realisiert werden. Dabei wurde die Erkennung auf ein rotes Stoppschild der Größe 8,5 cm $\times$ 8,5 cm trainiert. Anfangs ist mit einer einfachen Erkennung eines Stoppschildes gearbeitet worden, mit dem Ziel, bei der Erkennung des Stoppschilds den JetRacer zum Anhalten zu bringen. Dies konnte optimal umgesetzt werden, sodass im Anschluss eine abstandsabhängige Erkennung des Stoppschildes vollzogen wurde. Das heißt, hat das erkannte Stoppschild eine geringe Größe im Bild, bedeutet das, dass es noch weiter entfernt ist und die Geschwindigkeit des JetRacers nur reduziert werden muss. Wird das Stoppschild im Kamerabild groß, lässt das auf eine geringe Entfernung des JetRacers zu dem Stoppschild schließen und der JetRacer hält an. Hierbei ist darauf zu achten, dass bei der Anwendung immer eine gleiche Größe des zu erkennenden Stoppschilds von z. B. 8,5 cm $\times$ 8,5 cm genutzt wird, da es sonst zu Fehlinterpretationen kommt. Würde ein Stoppschild der Größe von 1 cm $\times$ 1 cm in dem

gleichen Abstand, wie ein Stoppschild der Größe von 5 cm $\times$ 5 cm platziert werden, würde das nur zu einer Reduzierung der Geschwindigkeit des JetRacers führen und nicht zu dem erforderlichen Stillstand.

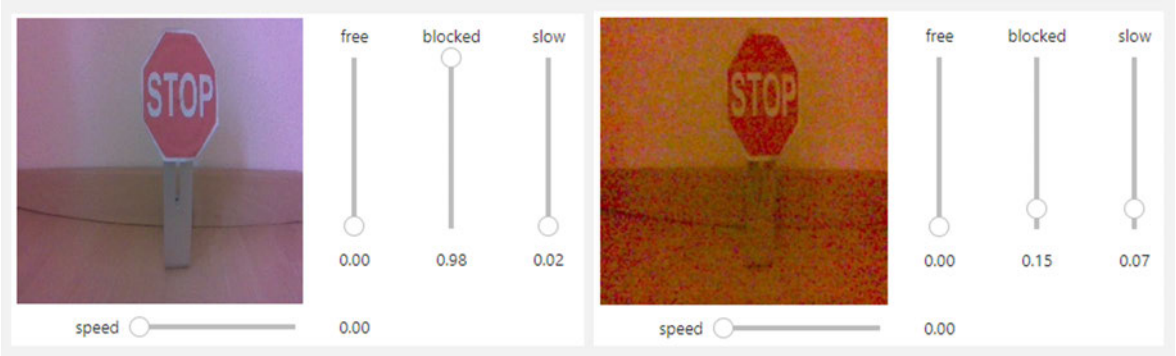


**Abbildung 4.6:** Grenzen der Hinderniserkennung bei schlechten Lichtverhältnissen

Des Weiteren wurde das Training des Modells vordergründig bei Tageslicht ohne künstliche Beleuchtung realisiert, was sich negativ auf die Objekterkennung des roten Stoppschildes bei Nutzung von künstlichem Licht mit erhöhtem Rotlicht-Anteil ausgewirkt hat. Das Ergebnis ist anhand der Abbildung 4.6 zu sehen, wo sich das Stoppschild und der JetRacer an exakt derselben Stelle befinden, jedoch in unterschiedlichen Ausleuchtungen. Das Stoppschild wird bei schwachen Lichtverhältnissen mitunter nicht erkannt, obwohl Form und Schrift des Stoppschilds weiterhin klar erkennbar sind. Das lässt darauf schließen, dass im zuvor trainierten Modell nicht nur die Form, sondern auch die Farbgebung trainiert wurde, was sich negativ auf die Anwendung bei schwachen Lichtverhältnissen auswirkt.

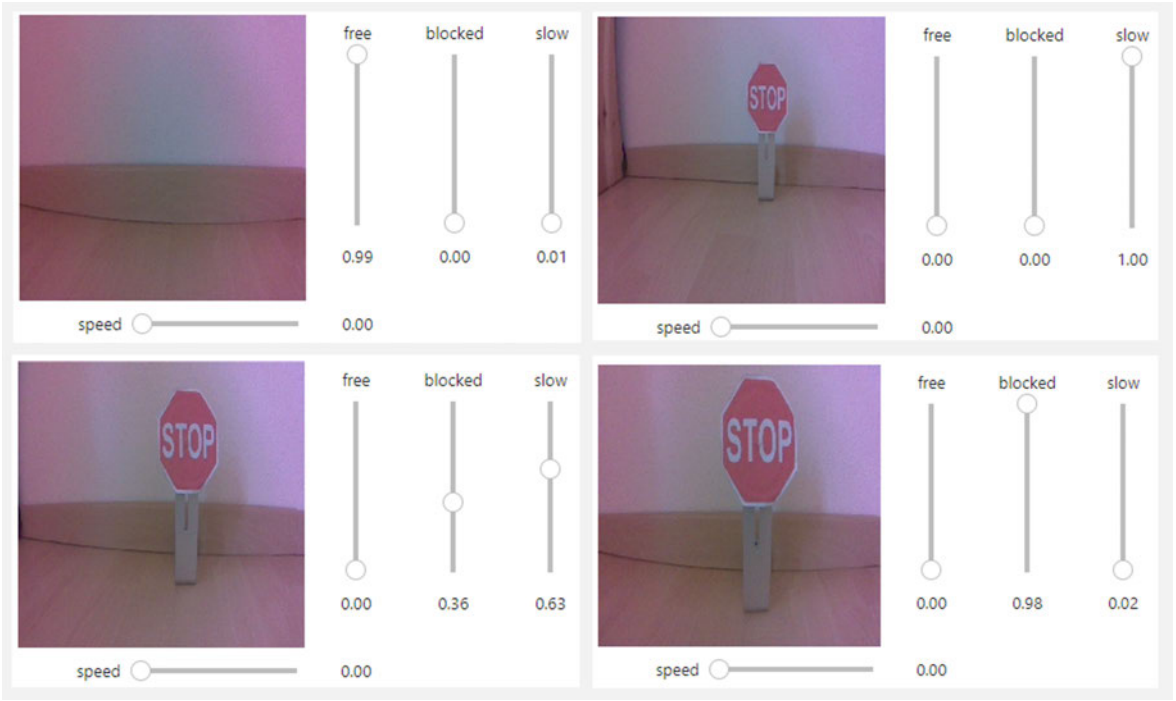


**Abbildung 4.7:** Beispiele für eine erfolgreiche Hinderniserkennung anhand eines Stoppschildes inkl. Wahrscheinlichkeitswerte für die verschiedenen Kategorien

Angesichts dessen wird dazu geraten, entweder bei verschiedenen Lichtverhältnissen das Stoppschild zu trainieren oder die Erkennung in den Grau-Wertebereich zu transferieren. Dadurch wird der Einfluss der Farbgebung eliminiert. Dies wurde innerhalb dieser Arbeit nicht weiterverfolgt, da Ursache und Lösung bekannt sind.

Anlässlich der erfolgreichen Objekterkennung des Stoppschild, wie anhand der Abbildung 4.7 dargestellt, soll innerhalb dieser Arbeit der Ansatz weiterverfolgt werden, das Stoppschild sich dynamisch bewegen zu lassen und damit eine Art Geschwindigkeitsregelung zu implementieren. Dazu wurde das Stoppschild an das Heck eines weiteren JetRacers montiert, um somit eine dynamische Abstanderkennung zu realisieren, die dazu führt, dass sich die Geschwindigkeit des folgenden JetRacers in Stufen anpasst. Auch diese Art der Umsetzung konnte erfolgreich realisiert werden, was als weiteres Etappenziel betrachtet werden kann, um eine „Follow-Me“-Funktion zu implementieren. Um die gewünschte „Follow-Me“-Funktion vollumfänglich umzusetzen, ist eine dynamische Objekterkennung zur Abstandsregulierung bei gleichzeitiger Spurverfolgung zu verfolgen, das im nächsten Abschnitt beschrieben wird.

## 4.3 Fusion von Spurverfolgung und Hinderniserkennung

Nachdem in den vorangegangenen Passagen erst die Spurverfolgung und anschließend die Objekterkennung zur dynamischen Geschwindigkeitsregelung bis zum Stillstand thematisiert worden ist, soll es in diesem Abschnitt darum gehen, beide Modelle, das der Spurverfolgung und das der Objekterkennung, zu fusionieren, damit beide Funktionen gleichzeitig auf dem JetRacer angewendet werden können.

### 4.3.1 Vorbereitung zur Fusion

Um gleichzeitig das Modell der Spurverfolgung und Objekterkennung in einem Notebook nutzen zu können, müssen die trainierten Modelle geladen werden. Dabei wird darauf hingewiesen, dass die Modelle bereits bestmöglich trainiert sein sollten, damit optimale Ergebnisse bei der gleichzeitigen Anwendung erzielt werden können. Die Modelle selbst werden über die Befehle,
`model.load_state_dict(torch.load('best_steering_model_xy.pth'))` und
`model.load_state_dict(torch.load('best_model_resnet18_3_Kat.pth'))` geladen.

Anschließenden sind, wie auch bereits zuvor in Unterabschnitt 4.1.3, die Vorverarbeitungsfunktionen für die Kamerabilder zu definieren, um die Bilder der Kamera in das Format, dass die Modelle verarbeiten können, zu transferieren. Des Weiteren werden die Daten für eine effiziente, parallele Bearbeitung an die GPU übertragen. Danach wird eine Instanz des JetRacers erstellt, um auf die Steuerung des JetRacers zugreifen zu können. Für eine Überprüfung der aktuellen Steuer- und Regelgrößen ist eine GUI umgesetzt worden, die anhand der Abbildung 4.8 dargestellt ist. Darin enthalten sind zum einen Steuergrößen für den JetRacer „JetRacer control items“, aber auch Parameter für die Hinderniserkennung „Collision avoidance control items“, sowie die aktuelle Sicht des JetRacers und die aus dem Modell vorhergesagten Werte für die Hinderniserkennung. Zuletzt wird noch der aktuelle Lenkwinkel dargestellt und die Möglichkeit, den JetRacer über eine Start- und Stopp-Taste zu steuern.

Nachdem die Benutzeroberfläche und die Vorbereitungen für die Fusion implementiert wurden, folgt im nächsten Kapitel die Beschreibung der Fusion und gleichzeitig die Anwendung auf dem JetRacer.

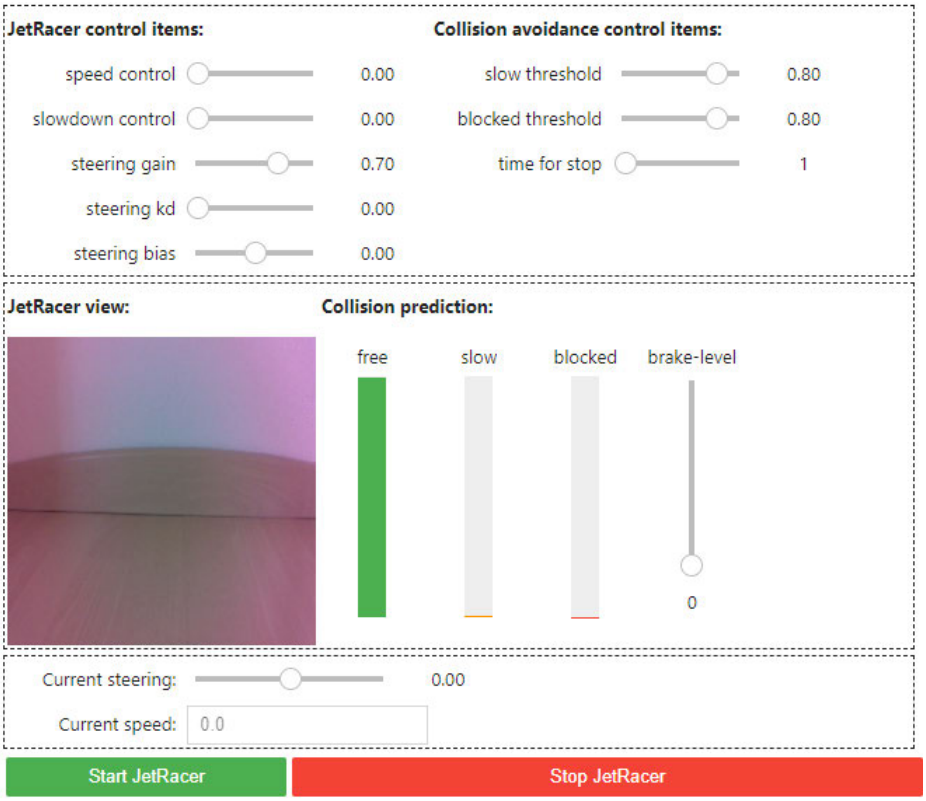


**Abbildung 4.8:** Abbildung des Benutzer-Interfaces für die Fusion von Spurverfolgung und Objekterkennung

### 4.3.2 Anwendung von Spurverfolgung und Objekterkennung

Bei der Verwendung des Modells zur Spurverfolgung gibt es keine Erweiterungen gegenüber der in Unterabschnitt 4.1.3 dargelegten Anwendung. Das Modell liefert XY-Koordinaten der Zielrichtung, anhand derer der Lenkwinkel berechnet wird. Die Nutzung der Objekterkennung wurde zu Beginn ebenfalls direkt aus dem Unterabschnitt 4.2.3 übernommen, um die Geschwindigkeit des JetRacers anhand der Modellvorhersage zu bestimmen. Die Wahrscheinlichkeiten der drei Kategorien „blockiert“, „langsam“ und „frei“ sind dabei verwendet worden. Der Befehl dazu ist in Zeile 1 von Quelltext 4.7 für die Kategorie „blockiert“ dargestellt.

```
1 prob_blocked = float(F.softmax(model_collision(image_preproc), dim=1).flatten
    ()[0])
```

**Quelltext 4.7:** Hinderniserkennung - Anwendung Auszug aus JupyterLab Notebook

Entsprechend den Wahrscheinlichkeiten der vorhergesagten Kategorien wird die Geschwindigkeit des JetRacers gesteuert. In Form einer Fallunterscheidung wird die Wahrscheinlichkeit der einzelnen Kategorien abgefragt. Ist die Wahrscheinlichkeit größer als 70 %, wird die dazugehörige Geschwindigkeit, die in der vorgestellten GUI für die Kategorien eingestellt wurde, übernommen.

Während der Anwendung stellte sich raus, dass diese drei Geschwindigkeitsstufen (Stillstand, langsam, maximale Geschwindigkeit) zu hart sind. Deswegen wurden Zwischenstufen, in Form von Bremsstufen bzw. Geschwindigkeitsstufen eingeführt. Somit gibt es nun fünf mögliche Stufen:

- Bremsstufe 0 - max. Geschwindigkeit entsprechend der Benutzereinstellung tritt ein, wenn die Wahrscheinlichkeit für die Kategorie „frei“ größer als 75 % ist und die Wahrscheinlichkeiten für „langsam“ und „blockiert“ kleiner als 25 %.

- Bremsstufe 1 - entspricht einer geringen Geschwindigkeitsreduktion, die sich einstellende Geschwindigkeit ist ein Mittelwert aus der vom Benutzer eingestellten Höchst- und Langsamen-Geschwindigkeit. Diese tritt ein, wenn die Wahrscheinlichkeit für die Kategorie „frei“ in dem Bereich von 10 % und 75 %, die Wahrscheinlichkeit für die Kategorie „langsam“ kleiner als 75 % und die Wahrscheinlichkeit für „blockiert“ kleiner als 25 % beträgt.
- Bremsstufe 2 - entspricht einer moderaten Geschwindigkeitsreduktion, wobei die Geschwindigkeit auf den Wert, der anhand der Abbildung 4.8 dargestellten GUI als Langsame- Geschwindigkeit definiert ist, herangezogen wird. Diese kommt zum Tragen, wenn die Wahrscheinlichkeit für die Kategorie „langsam“ mehr als 75 % beträgt und die Wahrscheinlichkeiten für die Kategorien „blockiert“ und „frei“ kleiner als 25 % sind.
- Bremsstufe 3 - stellt eine starke Geschwindigkeitsreduktion dar und ist die letzte Stufe, bevor der JetRacer zum Stillstand gebracht wird. Die sich einstellende Geschwindigkeit beträgt dabei 80 % der in der GUI eingestellten Langsamen-Geschwindigkeit.
- Bremsstufe 4 - bedeutet eine volle Geschwindigkeitsreduktion, sodass der JetRacer zum Stillstand kommt. Diese Stufe wird angewendet, wenn die Wahrscheinlichkeit der Kategorie „blockiert“ über 75 % beträgt und gleichzeitig die Wahrscheinlichkeiten für die Kategorien „langsam“ und „frei“ kleiner als 25 % sind.

Durch die Einführung der Brems- bzw. Geschwindigkeitsabstufungen kann ein flüssiges Fahrverhalten des JetRacers in Abhängigkeit des Abstandes zu einem erkannten Stoppschild geregelt werden. Über den Befehl,
`xy = model_trt(image_preproc).detach().float().cpu().numpy().flatten()`
erfolgt die Vorhersage der zu folgenden Spur aus dem Modell `best_model_resnet18_3_Kat.pth`. Dabei werden XY-Koordinaten durch das Modell vorhergesagt und für die Berechnung des Lenkwinkels des JetRacers genutzt, um der Spur zu folgen.

Damit ein kontinuierlicher Durchlauf mit jedem neuen Kamerabild durch die Modelle realisiert werden kann, muss der Befehl `camera.observe(execute, names='value')` ausgeführt werden. Dadurch wird der in diesem Abschnitt geschilderte Code mit jedem neuen Kamerabild ausgeführt. In dem nachfolgenden Abschnitt sollen die Ergebnisse der Fusion der beiden Funktionen dargelegt werden.

### 4.3.3 Ergebnisse der Fusion von Spurverfolgung und Hinderniserkennung

Nachdem die Funktionen erfolgreich auf dem JetRacer fusioniert werden konnten, wurde zum Testen im ersten Schritt die Spurverfolgung getestet. Es stellt sich dabei die Frage, ob diese im gleichen Maße funktioniert, wie bereits in der einzelnen Anwendung und in Unterabschnitt 4.1.3 beschrieben. Dabei wurde festgestellt, dass die Funktion der Spurverfolgung prinzipiell noch gegeben ist, jedoch eine starke Latenz von bis zu 5 s in der Verarbeitung der Bilddaten aufgetreten ist. Dieses Verhalten führt bei relativ geraden Strecken nicht direkt zu Fehlverhalten und kann noch akzeptiert werden. Bei einer kurvenreichen Strecke sind die Ergebnisse der Spurverfolgung hingegen nicht brauchbar, da eine Steuerung des Lenkwinkels dann erfolgt, wenn der JetRacer bereits die Spur verlassen und die zu folgende Spur aus dem Fokus verloren hat.

Ein ähnliches Verhalten konnte bei der Hinderniserkennung festgestellt werden. Dazu wurde ein Stoppschild auf der Spur platziert und getestet, ob der JetRacer die Geschwindigkeit in Kaskaden reduziert, bis er vor dem Stoppschild zum Stillstand kommt. Auch bei diesem Funktionsteil hat sich

die Latenz in der Verarbeitung negativ dargestellt und der JetRacer hat das Stoppschild überfahren. Erst nach mehreren Sekunden, nachdem die Bilder durch die Modelle verarbeitet worden, ist der JetRacer zum Stillstand gekommen.

Aufgrund der Tatsache, dass die Modelle einzeln zu akzeptablen Ergebnissen geführt haben und die Latenz in der Verarbeitung unter 1 s lagen, muss die Ursache in der nacheinander folgenden Verarbeitung der Kamerabilder in den Modellen zu dieser Latenz führen. Um die Latenz in der Verarbeitung zu minimieren, gilt es Optimierungen der Modelle zu prüfen, die zu einer schnelleren Verarbeitung der Bilder in den Modellen führt. Diese Optimierung wird in dem nachfolgenden Abschnitt betrachtet.

### 4.3.4 Optimierungen des Modelldurchsatzes innerhalb der Fusion

Die gleichzeitige Anwendung der Modelle für die Spurverfolgung und Hinderniserkennung innerhalb des PyTorch Framework hat gezeigt, dass diese Vorgehensweise prinzipiell funktioniert, jedoch aufgrund der hohen Latenz nicht anwendbar ist. Bei der Recherche nach Möglichkeiten, den Durchsatz innerhalb der Fusion der beiden Modell zu erhöhen, rückte das SDK Nvidia TensorRT in den Fokus. Bei den bisher verwendeten Modellen handelt es sich um PyTorch-Modelle. TensorRT ist eine Entwicklung von NVIDIA. Die Entwickler schreiben über diesen Optimierer, dass eine bis zu sechsmal schnellere Inferenz bei der Anwendung von TensorRT-Modellen anstatt von PyTorch-Modellen [46] erreicht werden kann. Des Weiteren wird auf der Webseite [47] geschrieben, dass die Anzahl an Bilder pro Sekunde (engl. Frames per Second) (fps) auf einen Jetson Nano mit einem ResNet-18 Modell von 29,4 fps auf 90,2 fps bei Nutzung eines TensorRT-Modells anstatt des PyTorch-Modells verbessert werden kann.

Torch-TensorRT Optimierer: Die Torch-TensorRT Bibliothek ist eine Integration für PyTorch, die die Inferenzoptimierungen von TensorRT auf NVIDIAs GPUs nutzt. Diese Integration nutzt die Vorteile der TensorRT-Optimierungen, wie 16-Bit-Fließkommazahl (engl. floating-point with 16 bit) (FP16) und 8-Bit-Ganzzahl (engl. Integer with 8 bit) (INT8) die eine reduzierte Genauigkeit bei jedoch höherem Durchsatz inkl. einer Rückfallebene zum nativen PyTorch bieten, wenn TensorRT die Modell-Subgraphen nicht unterstützt.

Dabei fungiert Torch-TensorRT als eine Erweiterung von TorchScript. Dieses optimiert und führt kompatible Teilgraphen aus und überlässt PyTorch die Ausführung des restlichen Graphen. PyTorchs umfassende und flexible Featuresets werden mit Torch-TensorRT verwendet, dass das Modell parst und Optimierungen auf die TensorRT-kompatiblen Teile des Graphen anwendet. Für weitere Details zu dem Optimierer und Konverter Torch2trt wird auf [48] verwiesen. Anhand der Abbildung 4.9 werden sechs mögliche Optimierungsansätze von TensorRT abgebildet.

Die vielversprechenden Ergebnisse innerhalb der Dokumentation von TensorRT haben zu der Entscheidung geführt, eine Optimierung der bestehenden PyTorch-Modelle für die NVIDIA GPUs mittels TensorRT durchzuführen. Die Schritte dazu werden nachfolgenden beschrieben.

Als Erstes müssen die bestehenden Modelle mittels des Konverters tensor2trt konvertiert werden. Dazu werden die Modelle samt Parameter geladen und an die GPU übertragen. Für die Erstellung des optimierten TensorRT-Modells wird ein Leer-Tensor benötigt, der später die Kamerabilddaten enthält. Mit dem Befehl `data = torch.zeros((1, 3, 224, 224)).cuda().half()` wird dieser Tensor, der anfangs nur aus Nullen besteht, erstellt.

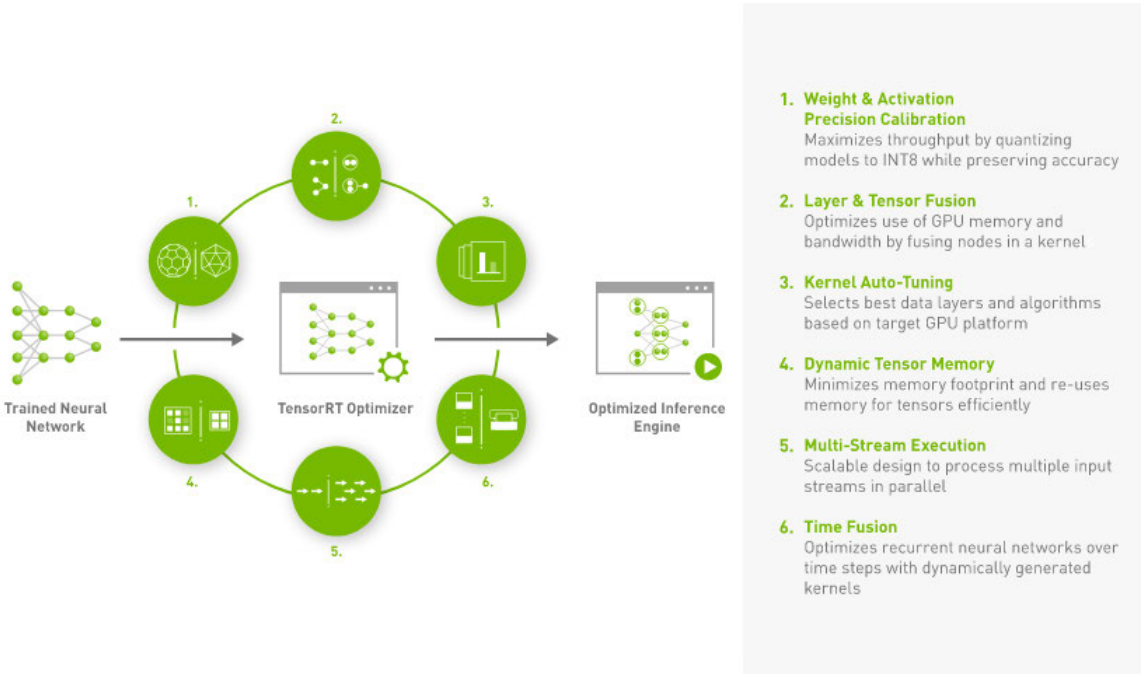


**Abbildung 4.9:** NVIDIA TensorRT – Modelldarstellung der Optimierungsmöglichkeiten eines neuronalen Netzes [46]

Mittels des Befehls `model_trt = torch2trt(model, [data], fp16_mode=True)` findet die Konvertierung von dem PyTorch-Modell zu dem für die NVIDIA GPUs optimierten TensorRT-Modell statt. Die Option `fp16_mode=True` reduziert die Genauigkeit. Anstatt Fließkommazahlen mit 32-Bit zu verarbeiten, werden sie mit 16-Bit verarbeitet. Die reduzierte Genauigkeit bei der Anwendung des Modells führt u. a. zu einem höheren Durchsatz, da die Berechnungen in der GPU mit halber Genauigkeit ausgeführt werden, wodurch die Berechnungen schneller werden.

Hierbei sei jedoch erwähnt, dass die Ausführung mit reduzierter Genauigkeit auch bereits in dem PyTorch-Modell mit der Funktion `.half()` genutzt wurde und dementsprechend im Vergleich keine Verbesserung in dem Durchsatz bietet.

Eine weitere Optimierungsmöglichkeit, um noch mehr Durchsatz innerhalb des Modells zu generieren, wäre dementsprechend auf den Datentyp INT8 zu wechseln. Dabei steht INT8 für Ganzzahlen mit einer Größe von 8-Bit. Diese Option stellt in Verbindung mit dem Jetson Nano keine Optimierung dar, da die GPU keine INT8 Berechnungen in optimierter Form unterstützt, sodass die Reduktion der Genauigkeit nicht mit einer Erhöhung des Durchsatzes in diesem Anwendungsfall verbunden ist.

TensorRT nutzt noch weitere Optimierungsschritte, um mehr Durchsatz innerhalb des Modells zu generieren. Ein Mittel dazu ist die optimierte Nutzung des GPU-Speichers und der Bandbreite durch Fusion von Knoten in einem Kernel oder der dynamische Tensorspeicher, der den Speicherplatzbedarf minimiert und Speicher für verwendete Tensoren wieder effizient nutzt. Weitere Optimierungen können wie bereits beschrieben in [48] nachgelesen werden.

Nachdem die vorhandenen Modelle in das TensorRT-Format konvertiert wurden und die Anwendung mittels dieser optimierten Modelle erneut ausgeführt wurde, siehe Unterabschnitt 4.3.2, konnte im direkten Vergleich eine Reduzierung der Latenz auf unter 1 s festgestellt werden, wodurch eine Anwendung bei geringer Geschwindigkeit des JetRacers möglich ist.

Bei höheren Geschwindigkeiten benötigt die Verarbeitung der Kamerabilder durch die Modelle weiterhin zu lang und der JetRacer verliert die Spur oder kann nicht rechtzeitig vor einem Hindernis zum Stillstand kommen.

Um die Latenz weiter zu optimieren, wurde der Ansatz verfolgt, auf die aktuellsten Bibliotheken und Framework-Versionen, die für den JetRacer bzw. den Jetson Nano verfügbar sind, zu aktualisieren. In den Versionshinweisen (engl. release notes) wird häufig auf Performance-Verbesserungen mit neuen Versionen hingewiesen. Um diese möglichen Potenziale zu heben, wurde von der bisher genutzten Nvidia JetPack-Version 4.5 für den Jetson Nano auf die aktuelle Software-Version 4.6.2 aktualisiert. [38]

Dabei ist zu erwähnen, dass es bereits eine JetPack-Versionen 5.0.2 gibt (Stand 28.11.2022), welche der Jetson Nano jedoch nicht unterstützt. Die JetPack-Version 4.6.2 nutzt eine Ubuntu Version als Betriebssystem der Version 18.04.6 LTS, das prinzipiell nur Python bis zur Version 3.6.9 unterstützt.

Aufgrund der Nutzung eines reinen JetPack-Images sind noch weitere Pakete zu installieren (z. B. pip, JupyterLab, traitlets), um eine Anwendung der Notebooks und Modelle zu ermöglichen. Essenziell für die Nutzung der Modelle ist dabei die Installation von PyTorch. Dabei kann maximal die Version PyTorch v1.10.0 genutzt werden, da diese nur eine Python-Version 3.6 erfordert. [49] Für die Installation von neueren PyTorch Versionen ist eine neuere Python Version erforderlich. Mittels des Terminal-Befehls,
`sudo -H pip3 install numpy torch-1.10.0-cp36-cp36m-linux_aarch64.whl`,
kann PyTorch in der Version 1.10.0 installiert werden. Anschließend ist die Installation von Torchvision erforderlich, die in Abhängigkeit der PyTorch und der Python Version installiert werden muss. Durch die PyTorch-Version 1.10.0 ist nur eine Installation der Torchvision Version 0.11.1 möglich, siehe [50]. Über diese sowie weitere kleiner Installationsschritte konnte das System erfolgreich auf die für den Jetson Nano aktuelle JetPack-Version 4.6.2 aktualisiert werden.

Die anschließende Anwendung der Fusion von Spurverfolgung und Hinderniserkennung resultierte anfänglich in einer Fehlermeldung, da die konvertierten TensorRT-Modelle mit einer älteren Version von PyTorch und TensorRT erstellt wurden. Es können jedoch nur die TensorRT-Modelle in der gleichen JetPack- und TensorRT-Version geladen werden, in der diese kompiliert wurden.

Dementsprechend musste die Konvertierung der ursprünglichen PyTorch-Modelle, die innerhalb der JetPack-Version 4.5 erstellt wurden, in der neuen Systemumgebung neu ausgeführt werden. Anschließend konnten die neuen TensorRT-Modelle in die Spurverfolgung und Hinderniserkennung eingebunden werden. Das Ergebnis innerhalb der Anwendung der fusionierten Modelle lieferte jedoch nicht den erwünschten Effekt und die Latenz innerhalb der Anwendung ist auf gleichem Niveau geblieben.

Eine weitere Möglichkeit der Optimierung liegt darin, die Modelle für die Spurverfolgung und Hinderniserkennung in der neuen PyTorch Umgebung auszuführen und anschließend die TensorRT-Modelle zu erzeugen. Dieser Weg wurde jedoch innerhalb der Arbeit nicht beschritten und ist Teil des Ausblicks, siehe Abschnitt 5.2.

## 4.4 Details und Handhabung der JupyterLab Notebooks

Nachdem in den vorangegangenen Abschnitten die Umsetzung der „Follow-Me“-Funktion im Vordergrund stand, soll in diesem Abschnitt der Umgang mit den JupyterLab Notebooks erläutert sowie Hilfsfunktion vorgestellt werden. Dabei wird auch auf Umsetzungen und Details eingegangen, die zur Bedienung des JetRacers und Visualisierung von z. B. aktuellen Steuergrößen beitragen.

Die JupyterLab Notebooks, die im Anhang als PDF-Export angefügt sind, sind analog der Arbeit in Spurverfolgung, Hinderniserkennung und Fusion der beiden Funktionen strukturiert und chronologisch auszuführen, damit die „Follow-Me“-Funktion umgesetzt wird. Die Notebooks können demzufolge als eine Art Anwendung betrachtet werden.

Das Notebook für die Spurverfolgung ist als Erstes zu öffnen und von oben nach unten auszuführen. Dabei ist das Notebook selbst wieder in die Abschnitte Datenerfassung, Training des Modells und Anwendung des Modells aufgeteilt.

Für die Datenerfassung, die bei der Spurverfolgung und der Hinderniserkennung benötigt wird, muss auf die Kamera des JetRacers zurückgegriffen werden. Die Kamera selbst wird mit dem Befehl aus Zeile 1 des Quelltextes 4.8 instanziiert. Für die Erfassung der Soll-Spur, die der JetRacer folgen soll, wird, wie anhand der Abbildung 4.10 dargestellt. Ein grüner Kreis in das Kamerabild gezeichnet, dessen Mittelpunkt die Koordinaten der Soll-Spur abbilden. Für die Darstellung des grünen Punktes innerhalb des aktuellen Kamerabildes wird die OpenCV Bibliothek genutzt. Der Befehl aus Zeile 14 des Quelltextes 4.8 lässt einen Kreis an die gesetzte XY-Koordinate mit dem Radius 8, der BGR-Farbe Grün und der Liniendicke 3 zeichnen.

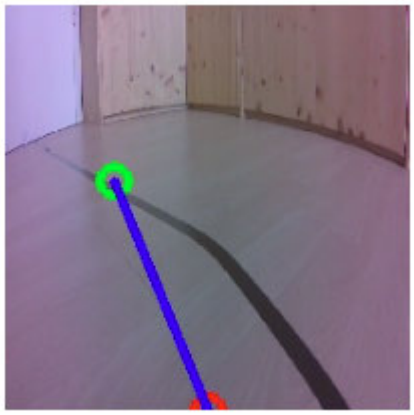

**Abbildung 4.10:** JetRacer Soll-Spurvorgabe

In dem JupyterLab Notebook zur Spurverfolgung wird das Bild für das Kamerabild mit der Soll-Spur, siehe Zeile 3 des Quelltextes 4.8, sowie die x- und y-Slider entsprechend Zeile 5 und 6 aus dem Quelltext 4.8 für die Koordinaten der Soll-Spur mittels des Widgets aus der ipywidgets-Bibliothek, siehe Zeile 24 aus Quelltext 4.8, in einer horizontalen Box dargestellt. Die XY-Koordinaten werden über Slider (Float-Slider) abgebildet, die ebenfalls in der ipywidgets-Bibliothek zur Verfügung gestellt werden. Dabei liegen die Koordinaten der Soll-Spur (Mittelpunkt des grünen Kreises) in der Mitte des Bildes, wenn die Slider in Null-Stellung sind, was durch Zeile 12 des Quelltextes 4.8 für die x-Koordinate realisiert wird. Analoges gilt für die y-Koordinate.

```
1 camera = CSICamera(width=224, height=224)
2
```

```
3  target_widget = widgets.Image(format='jpeg', width=widget_width, height=
       widget_height)
4
5  x_slider = widgets.FloatSlider(min=-1.0, max=1.0, step=0.001, description='x'
       )
6  y_slider = widgets.FloatSlider(min=-1.0, max=1.0, step=0.001, description='y'
       )
7
8  def display_xy(camera_image):
9      image = np.copy(camera_image)
10     x = x_slider.value
11     y = y_slider.value
12     x = int(x * widget_width / 2 + widget_width / 2)
13     y = int(y * widget_height / 2 + widget_height / 2)
14     image = cv2.circle(image, (x, y), 8, (0, 255, 0), 3)
15     image = cv2.circle(image, (int(widget_width / 2), widget_height), 8, (0,
       0,255), 3)
16     image = cv2.line(image, (x,y), (int(widget_width / 2), widget_height),
       (255,0,0), 3)
17     jpeg_image = bgr8_to_jpeg(image)
18     return jpeg_image
19
20 time.sleep(1)
21 traitlets.dlink((camera, 'value'), (image_widget, 'value'), transform=
       bgr8_to_jpeg)
22 traitlets.dlink((camera, 'value'), (target_widget, 'value'), transform=
       display_xy)
23
24 display(widgets.HBox([image_widget, target_widget]), x_slider, y_slider)
```

**Quelltext 4.8:** Auszug aus JupyterLab Notebook - Spurverfolgung - Datenerfassung

Die Koordinaten bzw. der Kreis kann dabei in dem Bild per Maus oder Gamepad platziert werden. In der Handhabung hat sich gezeigt, dass eine effektive Methode für die Datenerfassung das Gamepad, siehe Abbildung 4.11, darstellt. Es biete die Möglichkeit, den JetRacer fernzusteuern und gleichzeitig die Soll-Spur zu kennzeichnen, ohne die Hände von dem Eingabegerät zunehmen. Damit ist der Output innerhalb der Datenerfassung größer als durch Nutzung des Gamepads zur Steuerung des JetRacers und der Maus zum Kennzeichnen der Soll-Spur.

Bei der Nutzung des Gamepads ist zu beachten, dass es mehrere Betriebsmodi biete, wobei für die Anwendung hier auf den Modus zwei (analoger Modus) zurückgegriffen wird. Der Modus kann über die Select-Taste auf dem Gamepad, siehe Abbildung 4.11, eingestellt werden und wird über die darüber liegende LED-Anzeige dargestellt, wobei zwei leuchtende LEDs dem Modus zwei entsprechen. Somit können die Gamepad-Sticks in einem Bereich von [-1 . . . 0 . . . +1] und nicht wie im digitalen Modus nur als [-1, 0, 1] genutzt werden. Bei der Nutzung des Gamepads in Verbindung mit einem JupyterLab Notebook gibt es Browser, die die Funktionalitäten des Gamepads im Zusammenspiel mit JupyterLab nicht unterstützen. Eine Unterstützung konnte mit dem Microsoft Edge Browser, aber auch mit dem Google Chrome Browser gewährleistet werden. Keine Unterstützung bietet hingegen der Mozilla Firefox Browser.

Die Steuerung des JetRacers erfolgt dabei über den rechten Gamepad-Stick, siehe Zeile 4 und 5 des Quelltextes 4.9, während die Steuerung des Zielpfades mittels des linken Gamepad-Sticks erfolgt. Zum besseren Verständnis und Begriffserklärung des Gamepad-Sticks ist das Gamepad anhand der Abbildung 4.11 samt Markierung des linken Sticks dargestellt. Sobald ein Bild samt


**Abbildung 4.11:** Gamepad mit Markierung des linken Sticks [33]

Zielpfad aufgenommen werden soll, kann mittels des hinteren rechten oberen Buttons (Button 5) eine Speicherung des Bildes inklusive Zielpfad-Koordinaten erfolgen. Über den Befehl aus Zeile 7 des Quelltextes 4.9wird die linke horizontale Gamepad-Stick-Achse mit dem x-Slider verknüpft, sodass sich der Slider anpasst, sobald der Stick bewegt wird. Gleiches gilt für die y-Achse, siehe Zeile 8 des Quelltextes 4.9.

```
1 from jetracer.nvidia_racecar import NvidiaRacecar
2 jetracer = NvidiaRacecar()
3
4 steer_link = traitlets.dlink((controller.axes[2], 'value'), (jetracer, '
    steering'), transform=lambda x: -x)
5 throttle_link = traitlets.dlink((controller.axes[5], 'value'), (jetracer, '
    throttle'), transform=lambda x: x)
6
7 widgets.jsdlink((controller.axes[0], 'value'), (x_slider, 'value'))
8 widgets.jsdlink((controller.axes[1], 'value'), (y_slider, 'value'))
```

**Quelltext 4.9:** Auszug aus JupyterLab Notebook - Spurverfolgung - Steuerung des JetRacers

Mittels des Gamepads und der in Teilen vorgestellten Benutzeroberfläche ist es möglich, effizient die Datenerfassung für die Spurverfolgung abzuarbeiten. Im Anschluss sind die Abschnitte Training des Modells und Anwendung des Modells des JupyterLab Notebook Spurverfolgung auszuführen. Details dazu sind bereits in den Unterabschnitt 4.1.2 und Unterabschnitt 4.1.3 erläutert worden.

Anschließend ist das Notebook Hinderniserkennung zu öffnen und ebenfalls chronologisch auszuführen. Der Aufbau des Notebooks gleicht dabei dem der Spurverfolgung. Dabei wird an dieser Stelle auf Besonderheiten bei der Datenerfassung für die Hinderniserkennung eingegangen, die in dem Abschnitt 4.2 nicht erläutert wurden, da es sich dabei um Hilfsfunktionen handelt.

```
1 def save_snapshot(directory):
2     image_path = os.path.join(directory, str(uuid1()) + '.jpg')
3     with open(image_path, 'wb') as f:
4         f.write(image.value)
5
6 def save_free(change):
```

```
7     if change['new']:
8         global free_dir, free_count
9         save_snapshot(free_dir)
10        free_count.value = len(os.listdir(free_dir))
11
12 controller.buttons[7].observe(save_free, names='value')
13 controller.buttons[4].observe(save_slow, names='value')
14 controller.buttons[6].observe(save_blocked, names='value')
```

**Quelltext 4.10:** Auszug aus JupyterLab Notebook - Hinderniserkennung - Datenerfassung

Innerhalb des JupyterLab Notebook für die Hinderniserkennung wurde in dem Abschnitt der Datenerfassung eine GUI erstellt, die es dem Nutzer erleichtern soll, sich einen Überblick zu dem aktuellen Stand der Datenerfassung zu verschaffen. Dabei wurden drei verschiedene Tasten auf dem Gamepad den drei Kategorien (frei, langsam, blockiert) zugeordnet, siehe Quelltext 4.10 Zeile 12 - 14. Wird auf eine der drei Tasten (z. B. Taste 7) gedrückt, wird die Funktion aus Quelltext 4.10 Zeile 6 gestartet und das Bild mittels der Funktion aus Quelltext 4.10 Zeile 1 in das Verzeichnis für „frei (engl. free)“ abgespeichert. Dieses Vorgehen wird auch für die Fälle „langsam (engl. slow)“ und „blockiert (engl. blocked)“ angewendet.


**Abbildung 4.12:** Abbildung des Benutzer-Interfaces für die Datenerfassung innerhalb der Hinderniserkennung

Durch das Ausführen der beiden JupyterLab Notebooks Spurverfolgung und Hinderniserkennung werden die Modelle erstellt, die in dem letzten Notebook für die Fusion der Spurverfolgung und Hinderniserkennung benötigt werden. Auch dieses JupyterLab Notebook muss von oben nach unten ausgeführt werden. Zur Visualisierung der aktuellen Zustände des JetRacers wurde erneut eine GUI erstellt (siehe Abbildung 4.13). Dazu wurde auf bereits verwendete Elemente der ipywidgets-Bibliothek, wie Float-Slider, horizontale und vertikale Boxen und Buttons zurückgegriffen. Hinzu gekommen sind noch die FloatProgress Balken aus der ipywidgets-Bibliothek, um einen Status der aktuellen Hinderniserkennung farblich darzustellen, siehe Abbildung 4.13. Am Ende sollte der JetRa-

cer einer Spur folgen und einen Abstand zu einem vorausfahrenden JetRacer halten können, womit die Anwendung abgeschlossen ist. Es schließt sich im folgenden Kapitel eine Zusammenfassung der gesamten Arbeit an.

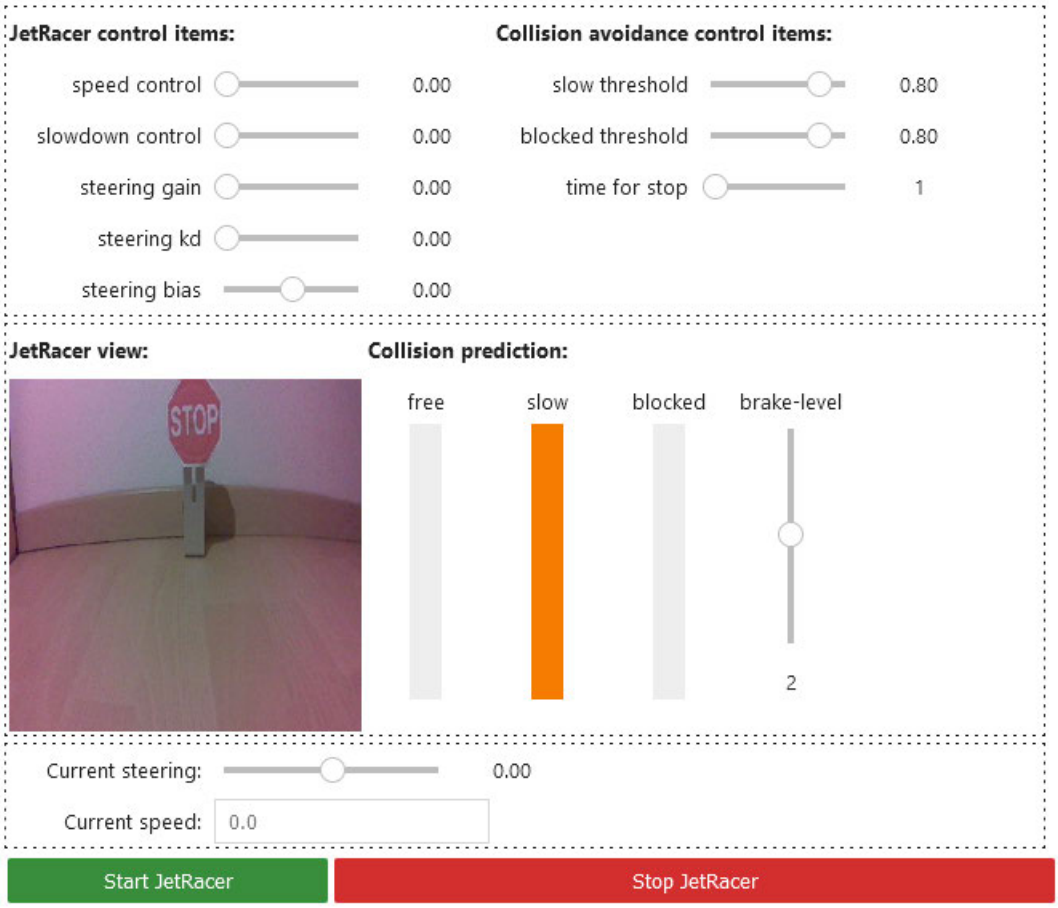


**Abbildung 4.13:** Abbildung des Benutzer-Interfaces innerhalb des JupyterLab Notebooks zur Fusion von Spurverfolgung und Hinderniserkennung

# 5 Zusammenfassung

Das Ziel der vorliegenden Arbeit war es, die Implementierung einer KI-gestützten „Follow-Me“-Funktion am Beispiel eines JetRacers. In der Arbeit wurde als Entwicklungsumgebung JupyterLab genutzt. Als Ergebnis sind drei Notebooks auf Python Basis geschrieben worden, die eine Anwendung auf dem JetRacer ermöglichen. Die Aufgabe teilte sich dabei in Spurverfolgung, Hinderniserkennung und die Fusion der beiden Teilmodelle auf.

Eine Vorgabe der Aufgabe war es, dass ein KI-Ansatz verfolgt werden soll. Das Gebiet der KI ist weitgefächert, daher wurde dieses im Rahmen der Arbeit näher dargelegt und eine konkrete Nutzung innerhalb der Arbeit spezifiziert. Für die Bearbeitung der Aufgabe wurde die Entscheidung getroffen, KNN zu nutzen, die durch überwachtes Lernen trainiert werden sollen. Dabei wurden CNN als spezielle Art der KNN genutzt, die innerhalb des Gebiets der KI und dem ML zum Teilgebiet des Deep Learning gehören.

Für die Spurverfolgung des Zielfahrzeugs wurde ein ResNet-18 Netz als CNN gewählt. Die gesamte Wertschöpfungskette, von der Datenerhebung, über das Training und der anschließenden Anwendung wurde innerhalb vom JupyterLab umgesetzt. Dabei wurden Bilder von der Spur in verschiedensten Situationen aufgenommen. Die Kennzeichnung der Spur erfolgt mittels Maus oder Gamepad. Die Bilder wurden zusammen mit der Information der zu folgenden Spur abgespeichert, wobei die Information zur Spur in Form von XY-Koordinaten in dem Dateinamen hinterlegt wurde.

Das Training des CNN verfolgt das Ziel, später bei unbekannten Kamerabildern einer Spur, der gefolgt werden soll, vorherzusagen. Das Training ist dabei auf Regression ausgelegt worden. Es erfolgte mittels der aufgenommenen Daten sowie den XY-Koordinaten der gekennzeichneten Spur.

Die Anwendung des Modells konnte zeigen, dass das Zielfahrzeug einer trainierten schwarzen Spur folgt. Grenzen sind dem System durch den eingeschränkten Blickwinkel, bei Kurvenfahrt oder zu hoher Geschwindigkeit und der verzögerten Aussage aus dem Modell, durch vorhandene Latenzen, aufgrund der Echtzeit Verarbeitung der Kameradaten innerhalb des Modells gegeben.

Im zweiten Schritt wurde eine Hinderniserkennung umgesetzt, um den zu folgenden JetRacer als solchen zu erkennen. Die Umsetzung ist analog der ersten Teilaufgabe in JupyterLab auf Python Basis erfolgt. Auch für diese Teilaufgabe sind eine Datenerfassung, Training eines CNN und eine abschließende Anwendung auf dem JetRacer umgesetzt worden. Für die Datenerfassung wurden die Bilder in die Kategorien „frei, langsam und blockiert“ kategorisiert, um verschiedene Abstände innerhalb der Anwendung des Modells vorhersagen zu können und entsprechend dem Abstand die Geschwindigkeit anzupassen. Das Training eines ResNet-18 Modells erfolgte dabei auf Klassifikation, um die Bilder der Kamera einer Klasse zuordnen zu können und in Abhängigkeit der Klasse die Geschwindigkeit zu regeln. Die Anwendung des trainierten CNN auf dem JetRacer war ebenfalls erfolgreich, sodass ein sich nah befindlicher JetRacer bzw. Stoppschild auch als solches erkannt wurde und der folgende JetRacer bis zum Stillstand abbremst.

Nach der erfolgreichen Umsetzung beider Teilaufgaben wurden die Modelle in einem eigenen Notebook fusioniert. Die Anwendung der beiden Modelle Spurverfolgung und Hinderniserkennung zur Abstandsregelung ist ebenfalls in JupyterLab auf Python-Basis umgesetzt worden. Die Funktion der Anwendung auf dem JetRacer, der einem Zielfahrzeug bzw. einem anderen JetRacer folgen soll, war anfangs nicht gegeben. Es bedurfte einer Optimierung der Modelle für die Spurverfolgung und

Hinderniserkennung. Diese Optimierung ist mittels de SDK TensorRT erfolgt, wodurch die vorhandenen PyTorch-Modelle in TensorRT-Modelle konvertiert wurden. Die konvertierten Modelle lieferten eine schnellere Verarbeitung der Kameradaten, wodurch die Latenz reduziert und eine gemeinsame Anwendung beider Modelle in einem JupyterLab Notebook realisiert werden konnte. Somit konnte mit Einschränkungen, vordergründig in Bezug auf die Geschwindigkeit des JetRacers, eine „Follow-Me“-Funktion mit einem KI-Ansatz auf einem JetRacer implementiert werden.

Im Folgenden werden die Ergebnisse aus den einzelnen Kapiteln zusammengefasst. Den Abschluss der Arbeit bildet ein Ausblick auf Themen, denen sich im Rahmen diese Arbeit nicht gewidmet werden konnte, die jedoch die Grundlage für zukünftige Untersuchungen bilden könnten.

## 5.1 Ergebnisse

**Stand der Technik** In der zum Anfang der Arbeit durchgeführten Recherche zum Stand der Technik wurden Fahrerassistenzsysteme und die dafür erforderlichen Techniken analysiert. Hieraus ergab sich, dass es aktuell nur einen OEM am Markt gibt, der autonomes Fahren nach SAE-Level 3 anbietet.

Weiterhin wurde überprüft, inwiefern KI-Ansätze im Automobilbereich eingesetzt werden. Es stellte sich raus, dass ML-Algorithmen, die dem Gebiet der KI zuzurechnen sind, z. B. für die Bildverarbeitung verwendet werden. Eine Sonderrolle spielt dabei der Hersteller Tesla, der bereits KNN für die Umsetzung für Fahrerassistenzsysteme nutzt.

Anschließend erfolgte ein Einblick in das Gebiet der KI. Der Schwerpunkt lag dabei auf dem ML und der darin enthaltenen KNN. Als Abschluss wurden auf eine Sonderform der KNN, die CNN, näher eingegangen.

**JetRacer** Die Umsetzung der „Follow-Me“-Funktion erfolgt unter Nutzung des JetRacers. In dem Abschnitt JetRacer wurden daher der Aufbau und die besondere Eignung des JetRacers, genauer gesagt des Nvidia Jetson Nano, vorgestellt.

Des Weiteren wurden die Bibliotheken, die für die Umsetzung der „Follow-Me“-Funktion mittels eines CNN erforderlich sind, dargelegt. Den Abschluss dieses Abschnittes stellt ein Einblick in die Probleme, die während der Nutzung des JetRacers aufgetreten sind, dar.

**Implementierung der Follow-Me Funktion** Der Hauptteil dieser Arbeit bildet die Implementierung der „Follow-Me“-Funktion auf den JetRacer. Dazu wurde die Aufgabe unterteilt und in diesen Teilabschnitten bearbeitet. Der erste Teil war die Umsetzung einer Spurverfolgung. Dazu wurde ein Resnet-18 CNN auf Regression trainiert, um die XY-Koordinaten anhand eines Kamerabildes der zu folgenden Spur vorherzusagen. Die Anwendung dieses Modell hat gezeigt, dass der JetRacer unter bestimmten Abstrichen einer Spur folgen kann. Anschließend wurde eine Hinderniserkennung implementiert, die es ermöglichen sollte, dass der JetRacer anhand von Kameradaten ein Hindernis erkennt und in Abhängigkeit des Abstandes des Hindernisses zu dem JetRacer die Geschwindigkeit dementsprechend anpasst. Für die Signalisierung eines Hindernisses wurde ein Stoppschild verwendet. Auch dafür wurde ein ResNet-18 CNN genutzt. Jedoch wurde dieses in dieser Teilaufgabe auf Klassifikation trainiert. Das Ziel, dass der JetRacer in Abhängigkeit des Abstandes der Hindernisse, im konkreten Fall des Stoppschilds, seine Geschwindigkeit anpasst, konnte erfolgreich implementiert werden. Dabei hat das CNN Netz die Bilder in Kategorien eingeteilt, „frei, langsam und blockiert“ und in Abhängigkeit dieser Kategorien bzw. der Wahrscheinlichkeit für das Auftreten einer dieser Kategorien wurde die Geschwindigkeit des JetRacers angepasst.

Für die vollumfängliche Umsetzung der „Follow-Me“-Funktion auf einen JetRacer wurden beide Modelle fusioniert. Dabei wurde jedoch festgestellt, dass die Verarbeitung der Kamerabilder durch die CNN, die nacheinander erfolgt, zu zeitintensiv ist und eine zu große Latenz entsteht, sodass eine Anwendung in dieser Form nicht zielführend war. Durch eine nachgelagerte Recherche konnte das SDK TensorRT ausfindig gemacht werden, das eine Optimierung der in PyTorch erstellten ResNet-18 Modelle zur Verfügung stellt. Mittels einer Konvertierung wurden die PyTorch ResNet-18 Modelle der Spurverfolgung und der Hinderniserkennung in ResNet-18 TensorRT-Modelle konvertiert. Die anschließende Anwendung der konvertierten Modelle stellt dahin gehend einen Erfolg dar, dass die Latenz auf unter 1 s reduziert und damit eine Anwendung auf dem JetRacer realisiert werden konnte.

Der JetRacer kann somit einem weiteren JetRacer folgen. Dabei verlässt er seine Spur nicht und steuert die Geschwindigkeit von Stillstand bis zur maximal eingestellten Geschwindigkeit in Stufen in Abhängigkeit erkannter Hindernisse. Es kann also von einer erfolgreichen Umsetzung der „Follow-Me“-Funktion mittels eines KI-Ansatzes auf dem JetRacer gesprochen werden.

Es schließt sich ein Ausblick auf Themen an, die im Rahmen der Arbeit nicht bearbeitet werden konnten, jedoch u. a. dazu beitragen könnten, das System und die Systemgrenzen positiv zu beeinflussen.

## 5.2 Ausblick

Nachdem die Implementierung der „Follow-Me“-Funktion anhand eines JetRacers erfolgreich umgesetzt werden konnte, wurde ein erster Schritt in Richtung des autonomen Fahrens unternommen. Die Nutzung der KNN in dieser Arbeit zeigt dabei, dass in dieser Herangehensweise noch große Potenziale stecken, um dem autonomen Fahren näherzukommen.

In weiteren Untersuchungen sollte sich daher darauf fokussiert werden, die Anwendung weiter zu optimieren und schließlich auszubauen. Das Ziel sollte dabei sein, einen teilautonomen oder später womöglich vollautonomen JetRacer implementieren zu können. Damit kann ein Teil dazu beigetragen werden, die aktuellen Probleme innerhalb des Individualverkehrs, wie die Überlastung der Infrastruktur oder zu hohe Emission zu lösen.

Im ersten Schritt sollte die Optimierung der vorhandenen Modelle zur Steigerung des Durchsatzes durch die Modelle, um eine weitere Reduzierung der Latenz zu erreichen, betrachtet werden. Hierzu bietet sich u. a. ein Vergleich des genutzten Framework PyTorch zu dem ebenfalls vorhandenen Framework TensorFlow an, um eine Entscheidung treffen zu können, welches Framework die beste Basis für kommende Arbeiten bildet. Des Weiteren sollte auch ein Vergleich möglicher Netzstrukturen erstellt werden. Aufgrund dessen, dass in [47] ein Vergleich geliefert wird, bei dem die Netze AlexNet oder Squeezenet einen höheren Durchsatz als ResNet-18 liefern.

Des Weiteren schließt sich eine Aktualisierung des JetRacers auf die aktuellste JetPack-Version 5.0.2 an, die jedoch offiziell nicht mehr von dem Jetson Nano unterstützt wird. Innerhalb von Recherchen ist jedoch die Webseite [51] gefunden worden, auf der beschrieben wird, wie der Jetson Nano trotzdem auf eine JetPack-Version 5.0.2 aktualisiert werden kann. Sollte die Aktualisierung erfolgreich sein, gilt es, zu überprüfen, ob die aktuellen Bibliotheken von PyTorch, Torchvision oder TensorRT zu einer besseren Performance in der Anwendung der Modelle führen können.

Einen weiteren Schwerpunkt neben der Optimierung der Anwendung stellt die Generierung von gekennzeichneten Daten für ein anschließendes Training dar. Hierbei gilt es, zu versuchen, eine automatische Kennzeichnung der Bilddaten zu realisieren, um im ersten Schritt für die Spurverfolgung, mehr Daten und damit besser Modelle generieren zu können. Im zweiten Schritt sollte die aktuelle Hinderniserkennung von einer Kategorisierung auf eine Regression umgestellt werden, um die Abstände zu einem Vorderfahrzeug nicht in Klassen, sondern dynamisch als Abstandwerte zu erhalten. Hierbei kann innerhalb der Objekterkennung auf vorhandene Implementierung von Begrenzungsrahmen (engl. bounding boxes) zurückgegriffen werden. Anhand der Breite eines solchen Begrenzungsrahmens kann der Abstand berechnet werden. Dazu kann zu Beginn die Breite des JetRacer manuell gemessen werden. Anschließend wird der vorausfahrende JetRacer in einem definierten und gemessen Abstand positioniert. Dieses Vorgehen stellt eine Art Kalibrierung dar, die unter Nutzung der Kameradaten des folgenden JetRacers eine Abstandsberechnung ermöglicht. Somit kann über die Anzahl der Pixel der Breite des Begrenzungsrahmens auf den Abstand des vorausfahrenden JetRacers geschlossen werden. Im weiteren Verlauf sollte die Kalibrierung in Abhängigkeit der erkannten Objekte erfolgen, so hat z. B. ein Pkw im Durchschnitt eine Breite von ca. 1,8 m (entspricht z. B. 50 Pixel), ein Lkw aber eine durchschnittliche Breite von 2,5 m (entspricht z. B. 60 Pixel), ein Fahrradfahrer 0,6 m usw. Somit ließe sich eine Abstandsberechnung in Abhängigkeit des erkannten Objekts realisieren, sowie eine dynamische Geschwindigkeitsregelung in Abhängigkeit des Abstandswertes zu dem Vorderfahrzeug.

Weitere Themen wären ein zyklisches Starten des Trainings anhand von automatisch generierten Trainingsdaten und ein automatisiertes Aktualisieren der trainierten Netze auf dem JetRacers.

Als langfristiges Ziel sollte die Umsetzung eines JetRacers der Stufe 4 nach SAE darstellen. Auf diesem Weg sind weitere Teilsysteme zu implementieren, wie ein Ausweich- oder Überhol-Assistent, der einen vorausfahrend JetRacer überholen kann.

An dieser Stelle gibt es eine Vielzahl von weiteren Funktionen, die auf dem JetRacer implementiert werden können, um das Ziel des voll-automatisierten oder gar autonomen JetRacers zu erreichen. Diese Arbeit bildet dazu einen ersten Baustein und zeigt, welche Potenziale in KNN stecken.

Anhang: JupyterLab - Notebook - Exporte

# Anlage 1: JupyterLab - Notebook - Spurverfolgung

Masterarbeit - Christian Schreiber - JetRacer - Spurerkennung mit Datenaufnahme mittels Gamepad

## 1 Spurverfolgung - Datenaufnahme

Um den JetRacer einer Spur folgen zu lassen, müssen folgende Schritte umgesetzt werden:

1. Datenaufnahme mit Labeling. Dazu muss der JetRacer in verschiedenen Positionen auf der Spur platziert werden. Innerhalb der live Darstellung muss der Zielpfad per Maus oder Gamepad ausgewählt werden. Der grüne Punkt im Bild symbolisiert dabei die Zielrichtung, wohin der JetRacer fahren soll. Jedes Bild wird dabei mit den X- und Y-Koordinaten des grünen Punktes abgespeichert

2. Modell mit den Daten trainieren. Im Abschnitt "Modell trainieren" wird ein neuronales Netz trainiert, um die X- und Y-Werte aus dem Labeling vorherzusagen. Somit kann der JetRacer einen ungefähren Lenkwinkel bestimmen, dabei handelt es sich nur um einen annähernden Winkel, da die Kamera nicht kalibriert ist

3. Modell mit den gelernten Gewichten auf das JetRacer übertragen

In diesem Notebook wird dabei nicht auf das Modell der Klassifikation z urückgegriffen, sondern auf die Regression.Für die Datenerfassung werden nachfolgende Bibliotheken benötigt, wobei hauptsächlich OpenCV verwendet wird, um Bilder mir Beschriftungen zu visualisieren und zu speichern

[1]:
```
# IPython Libraries for display and widgets
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display

# Camera and Motor Interface for JetBot
from jetracer.nvidia_racecar import NvidiaRacecar
from jetcam.csi_camera import CSICamera
from jetbot import bgr8_to_jpeg

# Basic Python packages for image annotation
from uuid import uuid1
import os
import glob
```

```
import numpy as np
import cv2
import time
```

Als Erstes muss das Model erstellt werden. Dabei muss das Model dem entsprechen, welches in dem interaktiven Trainingsnotebook erstellt wurde.

## 1.1 Darstellung des live Kamerabildes

Um ein live Bild zu erhalten, wird die Kamera als Erstes initialisiert. Das hier verwendete neurale Netz nutzt 224x224 Pixel Bilder als Eingang / Input. Damit nicht zu große Bilder zu verarbeiten sind, wird die Kameragröße auf diese Größe reduziert. Es wäre auch möglich, die Bilder in einem größeren Format aufzunehmen und anschließend zu reduzieren, das bedeutet aber weitere Rechenschritte.

[2]:
```
camera = CSICamera(width=224, height=224)
camera.running = True
widget_width = camera.width
widget_height = camera.height

image_widget = widgets.Image(format='jpeg', width=widget_width,␣
 ↪height=widget_height)
target_widget = widgets.Image(format='jpeg', width=widget_width,␣
 ↪height=widget_height)

x_slider = widgets.FloatSlider(min=-1.0, max=1.0, step=0.001, description='x')
y_slider = widgets.FloatSlider(min=-1.0, max=1.0, step=0.001, description='y')

def display_xy(camera_image):
    image = np.copy(camera_image)
    x = x_slider.value
    y = y_slider.value
    x = int(x * widget_width / 2 + widget_width / 2)
    y = int(y * widget_height / 2 + widget_height / 2)
    image = cv2.circle(image, (x, y), 8, (0, 255, 0), 3)
    image = cv2.circle(image, (int(widget_width / 2), widget_height), 8, (0,␣
 ↪0,255), 3)
    image = cv2.line(image, (x,y), (int(widget_width / 2), widget_height),␣
 ↪(255,0,0), 3)
    jpeg_image = bgr8_to_jpeg(image)
    return jpeg_image

time.sleep(1)
traitlets.dlink((camera, 'value'), (image_widget, 'value'),␣
 ↪transform=bgr8_to_jpeg)
```

```
traitlets.dlink((camera, 'value'), (target_widget, 'value'),␣
↪transform=display_xy)


display(widgets.HBox([image_widget, target_widget]), x_slider, y_slider)
```

```
FloatSlider(value=0.0, description='x', max=1.0, min=-1.0, step=0.001)
```

```
FloatSlider(value=0.0, description='y', max=1.0, min=-1.0, step=0.001)
```

## 1.2 Gamepad-Controller erstellen

Dieser Schritt wird genutzt, um, mittels des Gamepad den JetRacer und das Target zu steuern. Somit soll das Labeling einfacher gestaltet werden. Dabei wird der linke Gamepad-Stick zur Steuerung des Targets nutzen. Der rechte Gamepad-Stick soll zur Steuerung des JetRacers genutzt werden. Mittels einer Taste auf der Rückseite des Gamepads soll später das Bild mit der Target-Definition gespeichert werden.

Als Erstes muss eine Instanz des Controller-Widgets erstellt werden, mit dem, wie in einleitend erwähnt, die Bilder mit ”x”- und ”y”-Werten beschriften werden. Das Controller-Widget nimmt einen Index-Parameter entgegen, der die Nummer des Controllers angibt. Das ist für den Fall, das mehrere Controller angeschlossen sind, notwendig. Um den Index des Controllers zu ermitteln, kann wie folgt vorgegangen werden:

Öffnen der Webseite http://html5gamepad.com in einem neuen Browser-Tab oder Fenster (Achtung nur mit Chrome o. MS-Edge nutzbar, keine Firefox Unterstützung). Anschließend muss der Controller eingeschaltet werden. Sollte er bereits eingeschaltet sein, kann es sein, dass der Controller bei länger nicht Benutzung in den Standby geht. Der Controller, welcher in dem Waveshare-Kit mitgeliefert wurde, kann in diesem Fall über die ”Home”-Taste aufgeweckt werden. Auf der Webseite wird anschließend der Index des Controllers angegeben. Weiterhin kann die Funktion des Controllers überprüft werden.

Hinweis: Der von Waveshare mit gelieferte Controller hat verschiedene Betriebsmodi für die Controller-Sticks: - Digital (0 oder 1) oder → Mode 1 - Analog (0-1) → **Mode 2 - empfohlener Modus**

Der Mode kann über die Home-Taste auf dem Controller geändert werden und wird über die LED angezeigt, z.B. zwei LEDs leuchten bedeutet Mode 2.

Mit dem nun bekannten Index des Gamepads kann nun der Controller angelegt werden.

[3]:
```
controller = widgets.Controller(index=0)

display(controller)
```

```
Controller()
```

### 1.2.1 Gamepad-Controller zum Labeling verbinden

Das Gamepad ist jetzt angeschlossen, jedoch noch nicht zum Label der Bilder verbunden. Über die dlink-Funktion kann z.B. der linke Stick mit der x- und y-Achse in dem Target-Bild verbunden werden. Die Funktion dlink erlaubt eine Transformation zwischen Quelle und Ziel, welches die Funktion link nicht kann. Das bedeutet, dass die der Wert von den x- und y-Sildern keine Änderung an den Wert der Stick-Achsen hervorrufen kann.

[4]:
```
widgets.jsdlink((controller.axes[0], 'value'), (x_slider, 'value'))
widgets.jsdlink((controller.axes[1], 'value'), (y_slider, 'value'))
```

```
DirectionalLink(source=(Axis(value=-0.003921568393707275), 'value'),␣
↪target=(FloatSlider(value=0.0, descriptio…
```

### 1.2.2 Gamepad-Controller zur Steuerung des JetRacer verwenden

Für die einfachere Datenaufnahme wird der JetRacer über das Gamepad gesteuert, sodass der JetRacer bei der Datenaufnahme nicht per Hand neu platziert werden muss. Dazu wird der rechte Stick des Gamepads genutzt. Für die Positionierung des Zielpfades wird der linke Stick genutzt. Mittels Button 5 kann ein Bild gespeichert werden.

[5]:
```
from jetracer.nvidia_racecar import NvidiaRacecar
jetracer = NvidiaRacecar()

jetracer.throttle_gain = 0.2
jetracer.steering_offset=0
jetracer.steering = 0
left_link = traitlets.dlink((controller.axes[2], 'value'), (jetracer,␣
↪'steering'), transform=lambda x: -x) #Steering must be inverted so that the␣
↪axes of the gamepad stick correspond to the steering of the JetRacer
right_link = traitlets.dlink((controller.axes[5], 'value'), (jetracer,␣
↪'throttle'), transform=lambda x: x)
```

## 1.3 Datenaufnahme

Der nachfolgende Codeblock zeigt ein live Bild-Feed sowie die Anzahl der gespeicherten Bilder an. Es werden die X- und Y-Werte des grünen Kreises im Target (Zielbild) wie folgt gespeichert:

1. grünen Punkt auf das Ziel im Target-Bild platzieren
2. Drücken der Taste 1 auf der rechten Rückseite

Dadurch wird eine Datei im Ordner `dataset_xy` mit den Namen `xy_<x-Wert>_<y-Wert>_<uuid>.jpg` wobei `<x-Wert>` und `<y-Wert>` die Koordinaten in Pixel (nicht in Prozent) sind (von der linken oberen Ecke aus gezählt).

Beim später folgenden Model trainieren, werden auf dem Dateinamen der Bilder die x- und y-Werte extrahiert.

[6]:
```
DATASET_DIR = 'dataset_xy'

# "try/except" statement is needed because an error is returned if the␣
 ↪`dataset_xy` directory already exists.
try:
    os.makedirs(DATASET_DIR)
except FileExistsError:
    print('Verzeichnis wird nicht erstellt, da es bereits existiert.')

for b in controller.buttons:
    b.unobserve_all()

count_widget = widgets.IntText(description='count', value=len(glob.glob(os.path.
 ↪join(DATASET_DIR, '*.jpg'))))

def xy_uuid(x, y):
    return 'xy_%03d_%03d_%s' % (x * widget_width / 2 + widget_width / 2, y *␣
 ↪widget_height / 2 + widget_height / 2, uuid1())

def save_snapshot(change):
    if change['new']:
        uuid = xy_uuid(x_slider.value, y_slider.value)
        image_path = os.path.join(DATASET_DIR, uuid + '.jpg')
        with open(image_path, 'wb') as f:
            f.write(image_widget.value)
        count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

controller.buttons[5].observe(save_snapshot, names='value')

display(widgets.VBox([
    target_widget,
    count_widget
]))
```

```
Verzeichnis wird nicht erstellt, da es bereits existiert.
```

Kamera zurücksetzen, damit nachfolgende Anwendungen wieder auf die Kamera zurückgreifen können.

[ ]:
```
camera.running = False
camera.unobserve
```

**Hinweis:** Zum Abschluss der Datenerfassung muss der Kernel des Notebooks neu gestartet werden, damit die Kamera für spätere Anwendungen wieder freigegeben wird. Dazu per rechter Maustaste z.B. auf dieses Textfeld klicken und im Kontextmenü "Restart Kernel..." auswählen.
Damit sind auch alle bisher verwendeten Variablen zurückgesetzt, auch die Steuerung

des JetRacer über das Gamepad funktioniert nicht mehr.

# 2 Modell Trainieren

In diesen Abschnitt des Notebooks wird ein Modell trainiert, um anhand von gegebenen Eingangsbildern eine Reihe von x- und y-Werten auszugeben, die dem Ziel bzw. der Spur/ Straße entsprechen, der der JetRacer folgen soll.

Dazu wird das Deep-Learning-Framework PyTorch verwendet, um ein neuronales Modell vom Typ ResNet18 für die Spurverfolgung zu trainieren.

Zu Beginn müssen dazu wieder die relevanten Bibliotheken eingebunden werden.

[ ]:
```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
import glob
import PIL.Image
import os
import numpy as np
```

## 2.1 Datensatz-Instanz erstellen

Nun wird eine eigene `torch.utils.data.Dataset`-Implementierung erstellt, die die Funktionen `__len__` und `__getitem__` implementiert. Diese Klasse ist verantwortlich für das Laden von Bildern und das Parsen der x- und y-Werte aus den Dateinamen der Bilder. Durch die Implementierung der Klasse `torch.utils.data.Dataset`, können wir alle Torch Data Utilities verwenden.

Es sind einige Bildtransformationen (z.B. Farbflimmern) fest in dem Code umgesetzt, um die Datenbasis zu stärken. Zudem wurde eine zufällige horizontale Spiegelung optional umgesetzt, für den Fall, dass der JetRacer einer nicht symmetrischen Spur folgen soll, wie z.B. einer Straße, bei der JetRacer 'rechts bleiben' muss. Sollte das nicht von Bedeutung sein, ob der JetRacer einer bestimmten Konvention folgt, kann das Argument `random_hflips= true` gesetzt werden, damit wird der Datensatz noch mal erweitert.

[ ]:
```
def get_x(path, width):
    """Gets the x value from the image filename"""
    return (float(int(path.split("_")[1])) - width/2) / (width/2)

def get_y(path, height):
    """Gets the y value from the image filename"""
    return (float(int(path.split("_")[2])) - height/2) / (height/2)
```

```
class XYDataset(torch.utils.data.Dataset):

    def __init__(self, directory, random_hflips=False):
        self.directory = directory
        self.random_hflips = random_hflips
        self.image_paths = glob.glob(os.path.join(self.directory, '*.jpg'))
        self.color_jitter = transforms.ColorJitter(0.3, 0.3, 0.3, 0.3)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]

        image = PIL.Image.open(image_path)
        width, height = image.size
        x = float(get_x(os.path.basename(image_path), width))
        y = float(get_y(os.path.basename(image_path), height))

        if float(np.random.rand(1)) > 0.5:
            image = transforms.functional.hflip(image)
            x = -x

        image = self.color_jitter(image)
        image = transforms.functional.resize(image, (224, 224))
        image = transforms.functional.to_tensor(image)
        "mean, std = image.mean([1,2]), image.std([1,2])"
        image = image.numpy()[::-1].copy()
        image = torch.from_numpy(image)
        image = transforms.functional.normalize(image, [0.485, 0.456, 0.406],␣
↪[0.229, 0.224, 0.225])

        return image, torch.tensor([x, y]).float()

dataset = XYDataset('dataset_xy', random_hflips=False)
```

## 2.2 Aufteilung des Datensatzes in Trainings- und Testdatensatz

Nachdem der Datensatz eingelesen wurde, ist dieser in einen Trainings- und Testdatensatz aufzuteilen. Der Testsatz wird verwendet, um die Genauigkeit des trainierten Modells zu überprüfen.

In diesem Beispiel wird eine 90/10 Aufteilung gewählt. Das bedeutet 90 % des gesamten Datensatzes wird für das Training verwendet und die restlichen 10 % für das Testen.

[ ]:
```
test_percent = 0.1
num_test = int(test_percent * len(dataset))
train_dataset, test_dataset = torch.utils.data.random_split(dataset,␣
↪[len(dataset) - num_test, num_test])
```

Für die weitere Verarbeitung wird eine DataLoader erstellt, um die Daten Stapelweise (batches) zu verarbeiten. Dies reduziert die Speicherauslastung. In diesem Beispiel wird die Klasse `DataLoader`verwendet, um Daten in Stapeln/ Batches zu laden, Daten zu mischen und die Verwendung mehrerer Teilprozesse zu ermöglichen. Es wird eine Stapelgröße (batch_size) von 8 verwendet. Die Stapelgröße richtet sich nach dem auf Grafikprozessor verfügbaren Speicher und kann sich auf die Genauigkeit des Modells auswirken. Zur Veranschaulichung: Sind 1000 Trainingsbilder vorhanden und die Stapelgröße beträgt 500, dann werden 2 Iterationen benötigt, um eine Epoche abzuschließen.

[ ]:
```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)
```

## 2.3 Neuronales Netzmodell definieren

In diesem Notebook wird das ResNet-18-Modell, das auf PyTorch TorchVision verfügbar ist, verwendet. Dies Modell besitzt 18 Schichten / Layer

In einem Prozess, der Transfer-Lernen genannt wird, kann auf ein bereits trainiertes Modell (das auf Millionen von Bildern trainiert wurde) für eine neue Aufgabe verwendet werden, für die möglicherweise viel weniger Daten zur Verfügung stehen.

Weitere Einzelheiten zu ResNet-18: https://github.com/pytorch/vision/blob/master/..resnet.py

[ ]:
```
models.resnet18(pretrained=True)
```

Das ResNet-Modell hat eine vollständig verbundene (fc) letzte Schicht mit 512 Merkmalen als `in_features` und wird für die Regression trainiert, um eine Aussage zu den x- und y-Koordinaten zu machen. Dementsprechend hat das `out_features` die Größe 2 (für die x- und y-Koordinaten)

Das Modell erwarte normalisierte Eingangsbilder, d. h. Stapel von 3-Kanal-RGB-Bildern der Form (3 x H x W), wobei H und W mindestens 224 betragen sollten. Die Bilder müssen in einem Bereich von [0, 1] geladen und dann mit `mean = [0,485, 0,456, 0,406]` und `std = [0,229, 0,224,`

0,225] normalisiert werden. Eine Normalisierung hilft dabei Daten innerhalb eines bestimmten Bereiches zu erhalten, reduziert die Schiefe der Bilder und entfernt unnötige Informationen, wodurch das Training schneller abgeschlossen werden kann. Die Normalisierung reduziert auch die Gefahr von stark abnehmender oder zunehmender Gradienten. Die oben dargestellten Werte für den Mittelwert und die Standardabweichung sind die Werte des ImageNet Datensatzes und kann im Fall von stark zum ImageNet-Datensatz abweichenden Bildern durch `mean, std = image.mean([1,2]), image.std([1,2])` ersetzt werden.

Schließlich übertragen wir unser Modell zur Ausführung auf den Grafikprozessor

[ ]:
```
model.fc = torch.nn.Linear(512, 2)
device = torch.device('cuda')
model = model.to(device)
```

## 2.4 Regression trainieren

Das Modell wird im Beispiel auf 50 Epochen trainiert. Anschließend wird das Modell, wenn der Kostenfunktion optimiert / der Verlust reduziert wurde, abgespeichert.

Hinweis: Das Trainieren kann 30 - 60 Min. dauern.

[ ]:
```
NUM_EPOCHS = 50
BEST_MODEL_PATH = 'best_steering_model_xy.pth'
best_loss = 1e9

optimizer = optim.Adam(model.parameters())

for epoch in range(NUM_EPOCHS):

    model.train()
    train_loss = 0.0
    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.mse_loss(outputs, labels)
        train_loss += float(loss)
        loss.backward()
        optimizer.step()
    train_loss /= len(train_loader)

    model.eval()
    test_loss = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
```

```
        loss = F.mse_loss(outputs, labels)
        test_loss += float(loss)
    test_loss /= len(test_loader)

    print('%f, %f' % (train_loss, test_loss))
    if test_loss < best_loss:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_loss = test_loss
```

Sobald das Modell trainiert wurde, wird die Datei **best_steering_model_xy.pth** erzeugt, dieses Modell kann dann auf den JetRacer ausgerollt und für die Live-Versuche genutzt werden.

# 3 Spurverfolgung - Anwendung auf dem JetRacer

In diesem Abschnitt werden wir das zuvor trainierte Modell verwenden, um den JetRacer einer Spur folgen zu lassen.

## 3.1 Laden des trainierten Modells

Die Datei **best_steering_model_xy.pth** muss sich auf dem JetRacer befinden, dabei sollte die Datei in demselben Verzeichnis wie dieses Notizbuch sein. Die nachfolgenden Codezeilen initialisieren das PyTorch-Modell.

[ ]:
```
import torchvision
import torch

model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2)
```

Als Nächstes werden die trainierten Gewichte aus der Datei **best_steering_model_xy.pth** in das Modell geladen, dies kann ein paar Minuten dauern.

[ ]:
```
model.load_state_dict(torch.load('best_steering_model_xy.pth'))
```

Derzeit befinden sich die Modellgewichte im CPU-Speicher. Die nachstehenden Codezeilen, übertragen die Gewichte auf den Grafikprozessor. Mit dem Befehl **model.eval().half()** wird das Modell in den Evaluierungsmodus geschaltet, dabei wird die Genauigkeit durch die Option **half()** reduziert, was jedoch zu einem höheren Modell-Durchsatz führt.

[ ]:
```
device = torch.device('cuda')
model = model.to(device)
model = model.eval().half()
```

### 3.2 Erstellen der Vorverarbeitungsfunktion / Pre-Processing

Es wurde das trainierte Modell geladen, jedoch gibt es noch ein Problem, das Format, mit dem das Modell trainiert wurde, stimmt nicht genau mit dem Format der Kamera überein. Um das zu ändern, muss eine Bild-Vorverarbeitung durchgeführt werden. Dazu gehören die folgenden Schritte:

1. Konvertierung vom HWC-Layout zum CHW-Layout (H-Height, W-Width und C-Channel-Farbkanal)

2. Normalisierung mit denselben Parametern wie beim Training, die Kamera liefert Werte im Bereich von [0, 255] und die beim Training geladenen Bilder im Bereich [0, 1], sodass die Bilder mit 255.0 skaliert werden müssen.
3. Übertragen der Daten vom CPU-Speicher in den GPU-Speicher
4. Hinzufügen einer Batch-Dimension

[ ]:
```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()

def preprocess(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

Die Vorverarbeitungsfunktion ist somit definiert und die Bilder aus dem Kameraformat können in das Eingabeformat des neuronalen Netzes konvertieren werden. Als Nächstes wird die Kamera gestartet und das Kamerabild dargestellt.

> **Hinweis:** Sollte die Ausführung der nachfolgenden Codezeilen einen Fehler z.B. **"Could not initialize camera..."** bringen, so kann versucht werden, den Kernel per Rechtemaustaste neu zu starten und die Codezeilen ab 3. neu auszuführen. Oder im Terminal den Kameraservice über folgenden Befehl neu zu starten `!echo $USER | sudo -S systemctl restart nvargus-daemon`
>
> **Tipp:** zur bessern Übersichtlichkeit kann der Output des nachfolgenden Codeblock per Rechtsklick auf den Codeblock und der Option "Create New View for Output" in ein neues Sub-Fenster ausgegeben werden. Dabei besteht die Möglichkeit dieses Sub-Fenster den eigenen Ansprüchen zu verschieben*

[ ]:
```
from IPython.display import display
import ipywidgets
import traitlets
from jetbot import Camera, bgr8_to_jpeg
```

```
from jetcam.csi_camera import CSICamera

camera = CSICamera() #(width=224, height=224)
camera.running = True
image_widget = ipywidgets.Image()

traitlets.dlink((camera, 'value'), (image_widget, 'value'),␣
↪transform=bgr8_to_jpeg)

display(image_widget)
```

Des Weiteren wird eine JetRacer-Instanz für den Antrieb der Motoren des JetRacers benötigt.

[ ]:
```
from jetracer.nvidia_racecar import NvidiaRacecar

robot = NvidiaRacecar()
```

Jetzt werden Schieberegler zur Steuerung des JetRacer definiert > **Hinweis:** Die Werte der Schieberegler wurden mit den besten bekannten Konfigurationen initialisiert. Es ist jedoch möglich, dass diese Werte nicht der aktuellen Anwendung entsprechen. In diesem Fall sind die Schieberegler entsprechend anzupassen.

1. Geschwindigkeitskontrolle (speed_gain_slider): Um den JetRacer zu starten, muss der Wert vergrößert werden `speed_gain_slider`.
2. Lenkverstärkungssteuerung (steering_gain_slider): Sollte die Lenkung des JetRacer nervös arbeiten, kann der Schieberegler reduziert werden, bis die Lenkung sich beruhigt `steering_gain_slider`.
3. Lenkungs-Bias (steering_bias_slider): Sollte der JetRacer zu der rechten oder linken Seite der Strecke tendiert, kann dieser Schieberegler so eingestellt werden, dass JetRacer der Spur / Strecke in der Mitte folgt.

   **Hinweis:** Die Geschwindigkeitskontrolle erfolgt, anstatt über der Geschwindigkeitsregler über das Gamepad, um eine schnellere Reaktion des JetRacers zu gewährleisten.

   **Tipp:** zur bessern Übersichtlichkeit kann der Output des nachfolgenden Codeblock per Rechtsklick auf den Codeblock und der Option "Create New View for Output" in ein neues Sub-Fenster ausgegeben werden. Dabei besteht die Möglichkeit dieses Sub-Fenster den eigenen Ansprüchen zu verschieben*

[ ]:
```
speed_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01,␣
↪description='speed gain')
steering_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01,␣
↪value=0.7, description='steering gain')
steering_dgain_slider = ipywidgets.FloatSlider(min=0.0, max=0.5, step=0.001,␣
↪value=0.0, description='steering kd')
steering_bias_slider = ipywidgets.FloatSlider(min=-0.3, max=0.3, step=0.01,␣
↪value=0.0, description='steering bias')
```

```
display(speed_gain_slider, steering_gain_slider, steering_dgain_slider,␣
 ↪steering_bias_slider)
```

Als Nächstes werden die aktuellen JetRacer Werte, welche aus dem Modell berechnet werden, in Form von Schiebereglern dargestellt. Die Schieberegler x und y zeigen die voraussichtlichen x- und y-Werte an.

Der Schieberegler für die Lenkung zeigt den geschätzten Lenkwert an. Achtung dieser Wert entspricht nicht dem tatsächlichen Winkel des Ziels ist, sondern stellt einfach ein Wert, der nahezu proportional zu dem Lenkwinkel ist.

[ ]:
```
x_slider = ipywidgets.FloatSlider(min=-1.0, max=1.0, description='x')
y_slider = ipywidgets.FloatSlider(min=0, max=1.0, orientation='vertical',␣
 ↪description='y')
steering_slider = ipywidgets.FloatSlider(min=-1.0, max=1.0,␣
 ↪description='steering')
temp_slider = ipywidgets.FloatSlider(min=-90.0, max=90.0,␣
 ↪description='temp_slider')
speed_slider = ipywidgets.FloatSlider(min=0, max=1.0, orientation='vertical',␣
 ↪description='speed')

display(ipywidgets.HBox([y_slider, speed_slider]))
display(x_slider, steering_slider, temp_slider )
```

### 3.3 Steuerung der Geschwindigkeit des JetRacer über das Gamepad

Dazu ist wie bereits unter 1. erwähnt vorzugehen: 1. Besuchen der Webseite http://html5gamepad.com, um den Index des Gamecontroller zu bestimmen 2. Eintragen des Index in die nachfolgende Codezeile

[ ]:
```
import ipywidgets.widgets as widgets

controller = widgets.Controller(index=0)
display(controller)
```

Anschließend wird der Controller mit der Steuerung der Geschwindigkeitsregelung des JetRacer verbunden. Dabei wird der Geschwindigkeitsverstärkungsfaktor mit 0.2 recht klein gewählt, damit der JetRacer nicht zu zügig fährt. Eine Vergrößerung des Wertes erhöht die Geschwindigkeit des JetRacers.

> Hinweis: Sollte für die Fortbewegung des JetRacers eine andere Stick-Achse genutzt werden, so ist `controller.axes[5]` entsprechend der gewünschten Achse anzupassen

[ ]:
```
import traitlets
robot.throttle_gain = 0.2
right_link = traitlets.dlink((controller.axes[5], 'value'), (robot,␣
 ↪'throttle'), transform=lambda x: x)
```

Als Nächstes wird eine Funktion erstellt, die immer dann aufgerufen wird, wenn sich der Wert der Kamera ändert. Die Funktion führt dabei folgende Schritte aus:

1. Vorverarbeitung des Kamerabildes
2. Ausführen des neuronalen Netzwerks
3. Berechnung des ungefähren Lenkungswertes
4. Ansteuerung der Motoren

[ ]:
```
angle = 0.0
angle_last = 0.0

def execute(change):
    global angle, angle_last
    image = change['new']
    xy = model(preprocess(image)).detach().float().cpu().numpy().flatten()

    x = xy[0]
    y = (0.5 - xy[1]) / 2.0

    x_slider.value = x
    y_slider.value = y

    speed_slider.value = speed_gain_slider.value

    angle = np.arctan2(x, y)
    pid = angle * steering_gain_slider.value + (angle - angle_last) *␣
 ↪steering_dgain_slider.value
    angle_last = angle

    steering_slider.value = pid + steering_bias_slider.value
    temp_slider.value = angle
    robot.throttle = -speed_gain_slider.value
    robot.steering =  -steering_slider.value

execute({'new': camera.value})
```

Somit wurde eine Funktion zur Ausführung des neuronalen Netzes erstellt, welche jedoch noch mit der Kamera verbunden werden muss. Dies erfolgt über die `observe`-Funktion. > Warnung: Die nachfolgende Codezeile bewegt den JetRacer bzw. die Lenkung autonom und die Geschwindigkeitskontrolle wird über den Controller manuell gesteuert. Es ist sicherzustellen, dass der JetRacer sich in einem freien Raum befindet und auf der Spur/ Straße, auf der die Daten gesammelt wurden. Dabei ist die Spurverfolgung des neuronalen Netzes nur so gut wie die Daten, mit denen es trainiert wurde!

[ ]:
```
camera.observe(execute, names='value')
```

Wenn der JetRacer angeschlossen ist, sollte er jetzt mit jedem neuen Kamerabild neue Befehle erzeugen. Soll der JetRacer gestoppt werden, kann der Callback durch Ausführen des folgenden Codes deaktiviert werden.

[ ]:
```
import time

camera.unobserve(execute, names='value')

time.sleep(0.1)  # add a small sleep to make sure frames have finished␣
↪processing

camera.running = False
```

Die Kameraverbindung muss wieder ordnungsgemäß geschlossen werden, damit die Kamera in anderen Notebooks verwendet werden kann.

[ ]:
```
camera.running = False
camera.unobserve
```

**Hinweis:**
Zum Abschluss der live Demonstration muss der Kernel des Notebooks neu gestartet werden, damit die Kamera für spätere Anwendungen wieder freigegeben wird. Dazu per rechter Maustaste z.B. auf dieses Textfeld klicken und im Kontextmenü "Restart Kernel..." auswählen.
Damit sind auch alle bisher verwendeten Variablen zurückgesetzt, auch die Steuerung des JetRacer über das Gamepad funktioniert nicht mehr. Alternativ kann auch im Terminal folgender Befehl ausgefüllt werden: `!echo $USER | sudo -S systemctl restart nvargus-daemon`

# 4 Option Build TensorRT Modell für Spurverfolgung und Kollisionsvermeidung

## 4.1 Load the trained model

Die Datei `best_steering_model_xy.pth` muss in dem Ordner dieses Notizbuches liegen. Im nächsten Schritt werden die Bibliotheken geladen.

[ ]:
```
import torchvision
import torch

model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Lineavr(512, 2)
model = model.cuda().eval().half()
```

Als nächstes werden die trainierten Gewichte aus der Datei `best_steering_model_xy.pth`, geladen.

[ ]:
```
model.load_state_dict(torch.load('best_steering_model_xy.pth'))
```

Derzeit befinden sich die Modellgewichte im CPU-Speicher. Führen Sie den nachstehenden Code aus, um sie auf den Grafikprozessor zu übertragen.

[ ]:
```
device = torch.device('cuda')
```

## 4.2 Erzeugung des TensorRT Modells

`torch2trt` muss installiert sein, sollte das nicht der Fall sein, sind folgende Schritte in der Konsole auszuführen.

```
cd $HOME
git clone https://github.com/NVIDIA-AI-IOT/torch2trt
cd torch2trt
sudo python3 setup.py installieren
```

”torch2trt” konvertiert das Modell in ein TensorRT Modell, welches ein optimiertes Modell für schnellere Inferenz ist. Weiter Hinweise sind hier zu finden: torch2trt Readme.

Dieser Optimierungsprozess kann ein paar Minuten dauern.

[ ]:
```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)
```

Die folgende Zelle speichert das optimierte Modell.

[ ]:
```
torch.save(model_trt.state_dict(), 'best_steering_model_xy_trt.pth')
```

# Anlage 2: JupyterLab - Notebook - Kollisionsvermeidung

Masterarbeit - Christian Schreiber - JetRacer - Kollisionsvermeidung mit Datenaufnahme mittels Gamepad

## 1 Kollisionsvermeidung - Datenaufnahme

Ziel der Kollisionsvermeidung ist es, dass der JetRacer mit keinem Objekt kollidiert und vorher abbremst.
Dabei wird als Objekt, welches eine Kollision ankündigt ein Stoppschild genutzt, auf das der JetRacer trainiert wird.
Dieser Abschnitt erläutert die Datenaufnahme, um eine Kollisionsvermeidung mit dem JetRacer umzusetzen.

Dazu ist der JetRacer zuerst in die Positionen zu bringen, in denen keine Kollision droht und anschließend mit der CSICamera Bilder von der Umgebung Bilder zu machen. Ist das abgeschlossen, müssen Bilder von der Umgebung mit der drohenden Kollision gemacht werden.

### 1.1 Import der notwendigen Bibliotheken

[1]:
```
# IPython Libraries for display and widgets
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display

# Camera and Motor Interface for JetBot
from jetbot import Robot, Camera, bgr8_to_jpeg
from jetracer.nvidia_racecar import NvidiaRacecar
from jetcam.csi_camera import CSICamera
from jetracer.image import bgr8_to_jpeg


# Basic Python packages for image annotation
from uuid import uuid1
import os
import json
import glob
import datetime
```

```
import numpy as np
import cv2
import time
```

## 1.2 Gamepad-Controller erstellen

Dieser Schritt wird genutzt, um mittels des Gamepad den JetRacer und das Target zu steuern. Somit soll das Labeling einfacher gestaltet werden. Dabei wird der linke Gamepad-Stick zur Steuerung des Targets nutzen. Der rechte Gamepad-Stick soll zur Steuerung des JetRacers genutzt werden. Mittels einer Taste auf der Rückseite des Gamepads soll später das Bild mit der Target-Definition gespeichert werden.

Als Erstes muss eine Instanz des Controller-Widgets erstellt werden, mit dem, wie in einleitend erwähnt, die Bilder mit "x"- und "y"-Werten beschriften werden. Das Controller-Widget nimmt einen Index-Parameter entgegen, der die Nummer des Controllers angibt. Das ist für den Fall, dass mehrere Controller angeschlossen sind, notwendig. Um den Index des Controllers zu ermitteln, kann wie folgt vorgegangen werden:

Öffnen der Webseite http://html5gamepad.com in einem neuen Browser-Tab oder Fenster (Achtung nur mit Chrome o. MS-Edge nutzbar, keine Firefox Unterstützung). Anschließend muss der Controller eingeschaltet werden. Sollte er bereits eingeschaltet sein, kann es sein, dass der Controller bei länger nicht Benutzung in den Standby geht. Der Controller, welcher in dem Waveshare-Kit mitgeliefert wurde, kann in diesem Fall über die "Home"-Taste aufgeweckt werden. Auf der Webseite wird anschließend der Index des Controllers angegeben. Weiterhin kann die Funktion des Controllers überprüft werden.

Hinweis: Der von Waveshare mit gelieferte Controller hat verschiedene Betriebsmodi für die Controller-Sticks: - Digital (0 oder 1) oder → Mode 1 - Analog (0-1) → **Mode 2 - empfohlener Modus**

Der Mode kann über die Home-Taste auf dem Controller geändert werden und wird über die LED angezeigt, z.B. zwei LEDs leuchten bedeutet Mode 2.

Mit dem nun bekannten Index des Gamepads kann nun der Controller angelegt werden.

[2]:
```
controller = widgets.Controller(index=0)

display(controller)
```

```
Controller()
```

[3]:
```
jetracer = NvidiaRacecar()

jetracer.throttle_gain = 0.2
jetracer.steering_offset=0
jetracer.steering = 0
left_link = traitlets.dlink((controller.axes[0], 'value'), (jetracer,␣
↪'steering'), transform=lambda x: -x) #Steering must be inverted so that the␣
↪axes of the gamepad stick match the steering of the JetRacer.
```

```
right_link = traitlets.dlink((controller.axes[1], 'value'), (jetracer,␣
↪'throttle'), transform=lambda x: x)
```

### 1.2.1 live Kamera-Feed darstellen

In diesem Schritt wird die Kamera initialisiert und anschließend das Bild dargestellt.

> Das neuronale Netz benötigt ein Bild mit 224x224 Pixeln als Eingabe. Die Kamera wird auf diese Größe eingestellt, um die Dateigröße unseres Datensatzes zu minimieren.

[4]:
```
camera = CSICamera(width=224, height=224)
camera.running = True

image = widgets.Image(format='jpeg', width=224, height=224)  # this width and␣
↪height doesn't necessarily have to match the camera

camera_link = traitlets.dlink((camera, 'value'), (image, 'value'),␣
↪transform=bgr8_to_jpeg)

display(image)
```

```
Image(value=b'\xff\xd8\xff\xdb\x00C\
```

Nun werden Verzeichnisse benötigt, in denen alle gesammelten Daten gespeichert werden. Dazu werden die Ordner **dataset**, mit drei Unterordner **free**, **slow** und **blocked** erstellt, in denen die Bilder für jedes Szenario ablegt werden.

[5]:
```
blocked_dir = 'dataset_3_kategorien/blocked'
slow_dir = 'dataset_3_kategorien/slow'
free_dir = 'dataset_3_kategorien/free'

# "try/except" statement is used because these next functions can throw an␣
↪error if the directories exist already
try:
    os.makedirs(slow_dir)
    os.makedirs(free_dir)
    os.makedirs(blocked_dir)
except FileExistsError:
    print('Directories not created because they already exist')
```

```
Directories not created because they already exist
```

Der Jupyter-Dateibrowser sollte nach einer Aktualisierung die erstellten Verzeichnisse anzeigen. Weiterhin werden drei Schaltflächen erstellt inkl. Anzeigen, mit denen die Bilder für jedes Klassenlabel entsprechend gelabelt werden können. Außerdem werden Textfelder hinzugefügt, die anzeigen, wie viele Bilder von jeder Kategorie bisher gesammelt wurden. Ziel sollt es sein ebenso viele **free**, **slow** wie **blocked** Bilder zu sammeln. Es ist auch hilfreich zu wissen, wie viele Bilder insgesamt gesammelt wurden.

[6]:
```
button_layout = widgets.Layout(width='128px', height='64px')
free_button = widgets.Button(description='add free', button_style='success',␣
 ↪layout=button_layout)
blocked_button = widgets.Button(description='add blocked',␣
 ↪button_style='danger', layout=button_layout)
slow_button = widgets.Button(description='add slow', button_style='warning',␣
 ↪layout=button_layout)
free_count = widgets.IntText(layout=button_layout, value=len(os.
 ↪listdir(free_dir)))
blocked_count = widgets.IntText(layout=button_layout, value=len(os.
 ↪listdir(blocked_dir)))
slow_count = widgets.IntText(layout=button_layout, value=len(os.
 ↪listdir(slow_dir)))

display(widgets.HBox([free_count, free_button]))
display(widgets.HBox([slow_count, slow_button]))
display(widgets.HBox([blocked_count, blocked_button]))
```

```
HBox(children=(IntText(value=362, layout=Layout(height='64px', width='128px')),␣
 ↪Button(button_style='success',…
```

```
HBox(children=(IntText(value=329, layout=Layout(height='64px', width='128px')),␣
 ↪Button(button_style='warning',…
```

```
HBox(children=(IntText(value=397, layout=Layout(height='64px', width='128px')),␣
 ↪Button(button_style='danger', …
```

Die Schaltflächen müssen nun noch verknüpft werden, damit sie funktionieren. Dazu werden die `on_click`-Events der Schaltflächen genutzt. Hierbei wird der Wert des `Image`-Widgets (und nicht der Kamera) gespeichert, da die Bildinformationen im `Image`-Widget bereits im komprimierten JPEG-Format vorliegen.

Um sicherzustellen, dass sich keine Dateinamen wiederholen (auch nicht auf verschiedenen Rechnern!), wird das `uuid`-Paket in Python verwendet, welches die Methode `uuid1` enthält, um einen eindeutigen Bezeichner zu definieren. Dieser eindeutige Bezeichner wird aus Informationen wie der aktuellen Uhrzeit und der Rechneradresse generiert.

[7]:
```
for b in controller.buttons:
    b.unobserve_all()

def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)

def save_free(change):
    if change['new']:
        global free_dir, free_count
        save_snapshot(free_dir)
```

```
        free_count.value = len(os.listdir(free_dir))

def save_slow(change):
    if change['new']:
        global slow_dir, slow_count
        save_snapshot(slow_dir)
        slow_count.value = len(os.listdir(slow_dir))

def save_blocked(change):
    if change['new']:
        global blocked_dir, blocked_count
        save_snapshot(blocked_dir)
        blocked_count.value = len(os.listdir(blocked_dir))

# For control with the controller
controller.buttons[7].observe(save_free, names='value')
controller.buttons[4].observe(save_slow, names='value')
controller.buttons[6].observe(save_blocked, names='value')
```

### 1.2.2 Datenerfassung

Sammlung der Daten

1. JetRacer so in einem Szenario platzieren, in dem er blockiert ist und mit dem Label `add blocked` versehen
2. JetRacer in einem Szenario so platzieren, dass er frei ist und mit dem Label `add free` versehen
3. JetRacer in einem Szenario so platzieren, dass er in einem langsam fahrenden Zustand sein sollte und mit dem Label `add slow` versehen
4. Diese Schritte mehrfach wiederholen bis ausreichend Bilder gesammelt wurden ca. min. 100 pro Label

   Tipp: Das Widgets kann in ein neues Fenster per recht Maustaste verschoben werden in dem im Kontextmenü auf `Create New View for Output` geklickt wird

   Tipp für die Beschriftung von Daten:
   1. verschiedene Ausrichtungen des JetRacers verwenden 2. verschiedene Beleuchtungen des Umfelds nutzen 3. verschiedene Objekt-/Kollisionstypen nutzen, z.B. Wände, Vorsprünge, Objekte 4. verschiedene texturierte Böden/Objekte/Hintergründe nutzen

[8]:
```
display(image)
display(widgets.HBox([free_count, free_button]))
display(widgets.HBox([blocked_count, blocked_button]))
display(widgets.HBox([slow_count, slow_button]))
```

```
Image(value=b'\xff\x00\x01\x00\x01\x00\x00\xff\xdb\x00C\
```

```
HBox(children=(IntText(value=362, layout=Layout(height='64px', width='128px')),␣
↪Button(button_style='success',…
```

```
HBox(children=(IntText(value=397, layout=Layout(height='64px', width='128px')),␣
↪Button(button_style='danger', …
```

```
HBox(children=(IntText(value=329, layout=Layout(height='64px', width='128px')),␣
↪Button(button_style='warning',…
```

Kamera zurücksetzen, damit nachfolgende Anwendungen wieder auf die Kamera zurückgreifen können.

[9]:
```
camera.running = False
camera.unobserve
```

[9]:
```
<bound method HasTraits.unobserve of <jetcam.csi_camera.CSICamera object at
0x7f0bc33940>>
```

Sollte die Kamera sich nicht freigeben lassen, kann der nachfolgende Terminalbefehl helfen, um den Kamera-Service komplett neu zu starten:

[2]:
```
from getpass import getpass
!echo {getpass()} | sudo -S systemctl restart nvargus-daemon
```

```
 ......
[sudo] password for jetson:
```

# 2 Modell Trainieren

In diesen Abschnitt des Notebooks wird ein Modell trainiert (ResNet18), um anhand von den gesammelten und gelabelten Eingangsbildern eine Bildklassifizierung umzusetzen.

## 2.1 Import der notwendigen Bibliotheken

[1]:
```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
```

### 2.1.1 Datensatz-Instanz erstellen

Nun werden die Klassen aus dem `ImageFolder` Mittels des Paketes `torchvision.datasets` ausgelesen. Weiterhin wird eine Transformation aus dem Paket `torchvision.transforms` hinzugefügt, um die Daten für das Training vorzubereiten.

**Achtung:** In dem Datensatz dürfen ausschließlich Bilder enthalten sein. Des Weiteren dürfen auch in dem Ordner dataset... keine weiteren Ordner oder Dateien sein, als die, die trainiert werden sollen. Andernfalls kommt es beim Training zu einem Fehler folgender Art: cuda runtime error (710) : device-side assert triggered. Im Zweifelsfall über ein Terminal, die Ordner mit `ls -all` durchsuchen und versteckte Dateien löschen über `rm Dateiname` oder Verzeichnisse über `rm -r Verzeichnis`

[ ]:
```
!unzip -q dataset_3_kategorien.zip
```

[2]:
```
dataset = datasets.ImageFolder(
    'dataset_3_kategorien',
    transforms.Compose([
        transforms.ColorJitter(0.1, 0.1, 0.1, 0.1),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
)
```

## 2.2 Aufteilung des Datensatzes in Trainings- und Testdatensatz

Nachdem der Datensatz eingelesen wurde, ist dieser in einen Trainings- und Testdatensatz aufzuteilen. Der Testsatz wird verwendet, um die Genauigkeit des trainierten Modells zu überprüfen.

In diesem Beispiel wird eine 50/50 Aufteilung gewählt. Das bedeutet 50 % des gesamten Datensatzes wird für das Training verwendet und die restlichen 50 % für das Testen.

[3]:
```
train_dataset, test_dataset = torch.utils.data.random_split(dataset,␣
↪[len(dataset) - 50, 50])
```

Für die weitere Verarbeitung wird eine DataLoader erstellt, um die Daten Stapelweise (batches) zu verarbeiten Erstellen Sie Datenlader, um Daten stapelweise zu laden. Dies reduziert die Speicherauslastung. In diesem Beispiel wird die Klasse `DataLoader`verwendet, um Daten in Stapeln/ Batches zu laden, Daten zu mischen und die Verwendung mehrerer Teilprozesse zu ermöglichen. Es wird eine Stapelgröße von 8 verwendet. Die Stapelgröße richtet sich nach dem auf Grafikprozessor verfügbaren Speicher und kann sich auf die Genauigkeit des Modells auswirken.

[4]:
```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0,
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
```

```
    shuffle=True,
    num_workers=0,
)
```

### 2.2.1 Definieren des neuronalen Netzes

Das neuronale Netz, das wir trainieren werden, wird in diesem Block definiert. Das *torchvision*-Paket bietet eine Sammlung von bereits trainierten Modellen, die verwendet werden können.

In einem Prozess, der *Transfer-Lernen* genannt wird, können ein bereits trainiertes Modell (das auf Millionen von Bildern trainiert wurde) für eine neue Aufgabe verwendet werden, für die möglicherweise viel weniger Daten zur Verfügung stehen.

Wichtige Merkmale, die beim ursprünglichen Training des vortrainierten Modells gelernt wurden, können für die neue Aufgabe wiederverwendet werden. Wir werden das Modell `resnet18` verwenden.

[5]:
```
model = models.resnet18(pretrained=True)
```

Das resnet18-Modell wurde ursprünglich für einen Datensatz mit 1000 Klassenbezeichnungen trainiert, aber dieser vorliegende Datensatz hat nur drei Klassenbezeichnungen! Dies wird durch Ersetzen der letzten Schicht durch eine neue, untrainierte Schicht, die nur drei Ausgaben hat erreicht.

[6]:
```
model.fc = torch.nn.Linear(512, 3)
```

Schließlich übertragen wir unser Modell zur Ausführung auf den Grafikprozessor

[7]:
```
device = torch.device('cuda')
model = model.to(device)
```

## 2.3 Klassifikation trainieren

Mit dem folgenden Code wird das neuronale Netz 30 Epochen lang trainiert, wobei das Modell mit der besten Genauigkeit nach jeder Epoche gespeichert wird.

Eine Epoche ist ein vollständiger Durchlauf durch die Daten.

[8]:
```
NUM_EPOCHS = 30
BEST_MODEL_PATH = 'best_model_resnet18_3_Kat.pth'
best_accuracy = 0.0

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

for epoch in range(NUM_EPOCHS):

    for images, labels in iter(train_loader):
        # get the inputs
```

```
        images = images.to(device)
        labels = labels.to(device)
        # zero the parameter gradients
        optimizer.zero_grad()
        # predict classes using images from the training set
        outputs = model(images)
        # compute the loss based on model output and real labels
        loss = F.cross_entropy(outputs, labels)
        # backpropagate the loss
        loss.backward()
        # adjust parameters based on the calculated gradients
        optimizer.step()

    test_error_count = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        test_error_count += float(torch.sum(torch.abs(labels - outputs.
↪argmax(1))))

    test_accuracy = 1.0 - float(test_error_count) / float(len(test_dataset))
    print('%d: %f' % (epoch, test_accuracy))
    if test_accuracy > best_accuracy:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_accuracy = test_accuracy
```

```
0: 0.900000
1: 0.800000
2: 0.840000
3: 0.820000
4: 0.840000
5: 0.920000
6: 0.880000
7: 0.920000
8: 0.880000
9: 0.960000
10: 0.920000
11: 0.880000
12: 0.920000
13: 0.880000
14: 0.900000
15: 0.940000
16: 0.920000
17: 0.920000
18: 0.880000
19: 0.840000
```

```
20: 0.840000
21: 0.880000
22: 0.920000
23: 0.920000
24: 0.920000
25: 0.880000
26: 0.820000
27: 0.920000
28: 0.920000
29: 0.920000
```

# 3 Kollisionsvermeidung - Anwendung auf dem JetRacer

In diesem Abschnitt werden wir das zuvor trainierte Modell verwenden, um den JetRacer vor einem Hindernis stoppen zu lassen.

## 3.1 Laden des trainierten Modells

Die Datei `best_model_resnet18.pth` muss sich auf dem JetRacer befinden, dabei sollte die Datei in demselben Verzeichnis wie dieses Notizbuch sein. Die nachfolgenden Codezeilen initialisieren das PyTorch-Modell.

[1]:
```
import torchvision
import torch

model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 3)
```

Als Nächstes werden die trainierten Gewichte aus der Datei `best_model_resnet18_3_Kat.pth` in das Modell geladen, dies kann ein paar Minuten dauern.

[2]:
```
model.load_state_dict(torch.load('best_model_resnet18_3_Kat.pth'))
```

[2]:
```
<All keys matched successfully>
```

Derzeit befinden sich die Modellgewichte im CPU-Speicher. Die nachstehenden Codezeilen, übertragen die Gewichte auf den Grafikprozessor.

[3]:
```
device = torch.device('cuda')
model = model.to(device)
model = model.eval().half()
```

## 3.2 Erstellen der Vorverarbeitungsfunktion / Pre-Processing

Es wurde das trainierte Modell geladen, jedoch gibt es noch ein Problem, das Format, mit dem das Modell trainiert wurde, stimmt nicht genau mit dem Format der Kamera überein. Um das zu

ändern, muss eine Bild-Vorverarbeitung durchgeführt werden. Dazu gehören die folgenden Schritte:

1. Konvertierung vom HWC-Layout zum CHW-Layout (H-Height, W-Width und C-Channel-Farbkanal)

2. Normalisierung mit denselben Parametern wie beim Training, die Kamera liefert Werte im Bereich von [0, 255] und die beim Training geladenen Bilder im Bereich [0, 1], sodass die Bilder mit 255.0 skaliert werden müssen.
3. Übertragen der Daten vom CPU-Speicher in den GPU-Speicher

[1]:
```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()

#normalize = torchvision.transforms.Normalize(mean, std)

def preprocess(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

```
  File "C:\Users\chris\AppData\Local\Temp/ipykernel_10980/3380109179.py", line 0
    'normalize = torchvision.transforms.Normalize(mean, std)

SyntaxError: EOL while scanning string literal
```

Die Vorverarbeitungsfunktion ist somit definiert und die Bilder aus dem Kameraformat können in das Eingabeformat des neuronalen Netzes konvertiert werden.
Als Nächstes wird die Kamera gestartet und das Kamerabild dargestellt.

> **Hinweis:** Sollte die Ausführung der nachfolgenden Codezeilen einen Fehler z.B. **"Could not initialize camera..."** bringen, so kann versucht werden, den Kernel per Rechtemaustaste neu zu starten und die Codezeilen ab 3. neu auszuführen. Oder im Terminal den Kameraservice über folgenden Befehl neu zu starten `sudo -S systemctl restart nvargus-daemon` bzw. aus dem Notebook heraus über die nachfolgende Anweisung:

[14]:
```
from getpass import getpass
!echo {getpass()} | sudo -S systemctl restart nvargus-daemon
```

```
......
```

```
[sudo] password for jetson:
```

**Tipp:** zur bessern Übersichtlichkeit kann der Output des nachfolgenden Codeblock per Rechtsklick auf den Codeblock und der Option "Create New View for Output" in ein neues Sub-Fenster ausgegeben werden. Dabei besteht die Möglichkeit dieses Sub-Fenster den eigenen Ansprüchen zu verschieben*

[5]:
```
import traitlets
from IPython.display import display
import ipywidgets.widgets as widgets

from jetbot import bgr8_to_jpeg
from jetcam.csi_camera import CSICamera

camera = CSICamera(width=224, height=224)
camera.running = True

image = widgets.Image(format='jpeg', width=224, height=224)
blocked_slider = widgets.FloatSlider(description='blocked', min=0.0, max=1.0,␣
↪orientation='vertical')
free_slider = widgets.FloatSlider(description='free', min=0.0, max=1.0,␣
↪orientation='vertical')
slow_slider = widgets.FloatSlider(description='slow', min=0.0, max=1.0,␣
↪orientation='vertical')
speed_slider = widgets.FloatSlider(description='speed', min=0.0, max=0.5,␣
↪value=0.0, step=0.01, orientation='horizontal')

camera_link = traitlets.dlink((camera, 'value'), (image, 'value'),␣
↪transform=bgr8_to_jpeg)

display(widgets.VBox([widgets.HBox([image, free_slider, blocked_slider,␣
↪slow_slider]), speed_slider]))
```

```
VBox(children=(HBox(children=(Image(value=b'\xff\xd8\x00\x00\x01\
```

Des Weiteren wird eine JetRacer-Instanz für den Antrieb der Motoren des JetRacers benötigt.

[6]:
```
from jetracer.nvidia_racecar import NvidiaRacecar

jetracer = NvidiaRacecar()
```

### 3.3 Gamepad-Controller erstellen

Dieser Schritt wird genutzt, um, mittels des Gamepad den JetRacer zu steuern, sobald ein Hindernis in der Erscheinung tritt, soll der JetRacer stehen bleiben und nicht weiter fahren. Das Vorgehen zum Einbinden des Controllers ist dabei identisch, zu dem Vorgehen aus der Datenaufnahme unter 1.

1. Besuchen der Webseite http://html5gamepad.com, um den Index des Gamecontroller zu bestimmen
2. Eintragen des Index in die nachfolgende Codezeile

[7]:
```
controller = widgets.Controller(index=0)

display(controller)
```

```
Controller()
```

### 3.3.1 Gamepad-Controller mit JetRacer Steuerung verbinden

Anschließend wird der Controller mit der Steuerung der Geschwindigkeitsregelung des JetRacer verbunden. Dabei wird der Geschwindigkeitsverstärkungsfaktor mit 0.2 recht klein gewählt, damit der JetRacer nicht zu zügig fährt. Eine Vergrößerung des Wertes erhöht die Geschwindigkeit des JetRacers.

> Hinweis: Sollte für die Fortbewegung des JetRacers eine andere Stick-Achse genutzt werden, so ist `controller.axes[5]` entsprechend der gewünschten Achse anzupassen

[8]:
```
jetracer.throttle_gain = 0.2
jetracer.steering_offset=0
jetracer.steering = 0
left_link = traitlets.dlink((controller.axes[2], 'value'), (jetracer,␣
 ↪'steering'), transform=lambda x: -x) #Steering must be inverted so that the␣
 ↪axes of the gamepad stick match the steering of the JetRacer.
right_link = traitlets.dlink((controller.axes[5], 'value'), (jetracer,␣
 ↪'throttle'), transform=lambda x: x)
```

Als Nächstes wird eine Funktion erstellt, die immer dann aufgerufen wird, wenn sich der Wert der Kamera ändert. Die Funktion führt dabei folgende Schritte aus:

1. Vorverarbeitung des Kamerabildes
2. Ausführen des neuronalen Netzwerks
3. Wenn die Ausgabe des neuronalen Netzes anzeigt, dass eine Blockade vorhanden ist, stoppt der JetRacer, andernfalls fährt er mit der gewünschten Controller Geschwindigkeit

[9]:
```
import torch.nn.functional as F
import time

def update(change):
    global blocked_slider, free_slider, slow_slider, robot
    x = change['new']
    x = preprocess(x)
    y = model(x)

    # we apply the `softmax` function to normalize the output vector so it sums␣
 ↪to 1 (which makes it a probability distribution)
```

```
    y = F.softmax(y, dim=1)

    prob_blocked = float(y.flatten()[0])
    prob_free = float(y.flatten()[1])
    prob_slow = float(y.flatten()[2])

    blocked_slider.value = prob_blocked
    free_slider.value = prob_free
    slow_slider.value = prob_slow

    if prob_blocked < 0.3:
        jetracer.throttle_gain = 0.2

    else:
        jetracer.throttle_gain = 0.0

    time.sleep(0.001)

update({'new': camera.value})  # we call the function once to initialize
```

Somit wurde eine Funktion zur Ausführung des neuronalen Netzes erstellt, welche jedoch noch mit der Kamera verbunden werden muss. Dies erfolgt über die `observe`-Funktion.

[10]:
```
camera.observe(update, names='value')  # this attaches the 'update' function to␣
↪the 'value' traitlet of the camera
```

Wenn der JetRacer angeschlossen ist, sollte er jetzt mit jedem neuen Kamerabild neue Befehle erzeugen. Soll der JetRacer gestoppt werden, kann der Callback durch Ausführen des folgenden Codes deaktiviert werden.

[13]:
```
import time

time.sleep(0.1)  # add a small sleep to make sure frames have finished␣
↪processing
camera.running = False
```

**Hinweis:**
Zum Abschluss der live Demonstration muss der Kernel des Notebooks neu gestartet werden, damit die Kamera für spätere Anwendungen wieder freigegeben wird. Dazu per rechter Maustaste z.B. auf dieses Textfeld klicken und im Kontextmenü "Restart Kernel..." auswählen.
Damit sind auch alle bisher verwendeten Variablen zurückgesetzt, auch die Steuerung des JetRacer über das Gamepad funktioniert nicht mehr. Alternativ kann auch im Terminal folgender Befehl ausgefüllt werden: `!echo $USER | sudo -S systemctl restart nvargus-daemon`

# 4 Option Build TensorRT Modell für Spurverfolgung und Kollisionsvermeidung

Die Datei `best_model_resnet18_3_Kat.pth` muss in dem Ordner dieses Notizbuches liegen. Im nächsten Schritt werden die Bibliotheken geladen.

[1]:
```
import torchvision
import torch

model = torchvision.models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 3)
model = model.cuda().eval().half()
```

Als Nächstes werden die trainierten Gewichte aus der Datei `best_model_resnet18.pth`, geladen.

[2]:
```
model.load_state_dict(torch.load('best_model_resnet18_3_Kat.pth'))
```

[2]:
```
<All keys matched successfully>
```

Derzeit befinden sich die Modellgewichte im CPU-Speicher. Führen Sie den nachstehenden Code aus, um sie auf den Grafikprozessor zu übertragen.

[3]:
```
device = torch.device('cuda')
```

## 4.1 Erzeugung des TensorRT Modells

`torch2trt` muss installiert sein, sollte das nicht der Fall sein, sind folgende Schritte in der Konsole auszuführen.

```
cd $HOME
git clone https://github.com/NVIDIA-AI-IOT/torch2trt
cd torch2trt
sudo python3 setup.py installieren
```

"torch2trt" konvertiert das Modell in ein TensorRT Modell, welches ein optimiertes Modell für schnellere Inferenz ist. Weiter Hinweise sind hier zu finden: torch2trt readme.

Dieser Optimierungsprozess kann ein paar Minuten dauern.

[4]:
```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)
```

Die folgende Zelle speichert das optimierte Modell.

[5]:
```
torch.save(model_trt.state_dict(), 'best_model_resnet18_3_Kat_trt.pth')
```

# Anlage 3: JupyterLab - Notebook - Fusion Spurverfolgung und Kollisionsvermeidung

Masterarbeit - Christian Schreiber - JetRacer - Spurverfolgung mit Follow-Me Funktion

## 1 Spurverfolgung mit Follow-Me Funktion auf dem JetRacer

In diesem Notebook werden sowohl ein optimiertes Regressions- als auch Klassifikationsmodell in einem Notizbuch kombiniert, sodass der JetRacer in der Lage ist, der Spur zu folgen (*Spurverfolgung*), als auch Kollisionen mit Hindernissen, die sich ihm in Echtzeit in den Weg stellen, zu vermeiden (*Kollisionsvermeidung*).

Zunächst wird der Gerätespeicher von der CPU in die GPU verlagert.

[1]:
```
import torch
device = torch.device('cuda')
```

### 1.1 Spurverfolgung

- Ablegen der Modelldatei "*best_steering_model_xy_trt.pth*" aus dem Notebook "Masterarbeit_Spurverfolgung_GamePad.ipynb" in das Verzeichnis des aktuellen Notebooks hoch.

### 1.2 Kollisionsvermeidung

- Ablegen der Modelldatei "*best_model_trt.pth*" in das Verzeichnis dieses Notebooks

Laden der optimierten TRT-Modelle, indem folgende Zelle ausgefürt wird

[2]:
```
import torch
from torch2trt import TRTModule

model_trt = TRTModule()
model_trt.load_state_dict(torch.load('best_steering_model_xy_trt.pth')) #␣
 ↪trainiertes Spurverfolgungsmodell

model_trt_collision = TRTModule()
model_trt_collision.load_state_dict(torch.load('best_model_resnet18_3_Kat_trt.
 ↪pth')) # trainiertes Kollisionsvermeidungsmodell
```

[2]:
```
<All keys matched successfully>
```

### 1.3 Erstellen der Vorverarbeitungsfunktion / Pre-Processing

Es wurde das trainierte Modell geladen, jedoch gibt es noch ein Problem, das Format, mit dem das Modell trainiert wurde, stimmt nicht genau mit dem Format der Kamera überein. Um das zu ändern, muss eine Bild-Vorverarbeitung durchgeführt werden. Dazu gehören die folgenden Schritte:

1. Konvertierung vom HWC-Layout zum CHW-Layout (H-Height, W-Width und C-Channel-Farbkanal)

2. Normalisierung mit denselben Parametern wie beim Training, die Kamera liefert Werte im Bereich von [0, 255] und die beim Training geladenen Bilder im Bereich [0, 1], sodass die Bilder mit 255.0 skaliert werden müssen.
3. Übertragen der Daten vom CPU-Speicher in den GPU-Speicher
4. Hinzufügen einer Batch-Dimension

[3]:
```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()

def preprocess(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

Die Vorverarbeitungsfunktion ist somit definiert und die Bilder aus dem Kameraformat können in das Eingabeformat des neuronalen Netzes konvertiert werden. Als Nächstes wird die Kamera gestartet und das Kamerabild dargestellt.

> **Hinweis:** Sollte die Ausführung der nachfolgenden Codezeilen einen Fehler z.B. **"Could not initialize camera..."** bringen, so kann versucht werden, den Kernel per Rechtemaustaste neu zu starten und die Codezeilen ab 3. neu auszuführen. Oder im Terminal den Kameraservice über folgenden Befehl neu zu starten `!echo $USER | sudo -S systemctl restart nvargus-daemon`
>
> **Tipp:** zur bessern Übersichtlichkeit kann der Output des nachfolgenden Codeblock per Rechtsklick auf den Codeblock und der Option "Create New View for Output" in ein neues Sub-Fenster ausgegeben werden. Dabei besteht die Möglichkeit dieses Sub-Fenster den eigenen Ansprüchen zu verschieben*

[ ]:
```
from getpass import getpass
!echo {getpass()} | sudo -S systemctl restart nvargus-daemon
```

[4]:
```
from IPython.display import display
import ipywidgets
import traitlets

from jetbot import bgr8_to_jpeg

from jetcam.csi_camera import CSICamera
camera = CSICamera(width=224, height=224, fps=10)
camera.running = True
```

[5]:
```
image_widget = ipywidgets.Image()

traitlets.dlink((camera, 'value'), (image_widget, 'value'),␣
 ↪transform=bgr8_to_jpeg)
```

[5]:
```
<traitlets.traitlets.directional_link at 0x7ef84777f0>
```

Des Weiteren wird eine JetRacer-Instanz für den Antrieb der Motoren des JetRacers benötigt.

[6]:
```
from jetracer.nvidia_racecar import NvidiaRacecar

robot = NvidiaRacecar()
```

Als Nächstes werden einige Schieberegler definiert, um den JetRacer zu steuern: > **Hinweis**: Die Werte der Schieberegler wurden für die beste bekannte Konfigurationen initialisiert, diese könnten jedoch für andere Datensätze nicht funktionieren, daher sind die Schieberegler entsprechend der eigenen Einrichtung und Umgebung anzupassen.

1. Schieberegler Geschwindigkeitskontrolle: Um den JetRacer zu starten, ist der `speed_control_slider` - Silder zu vergrößern
2. Schieberegler "Steering Gain": Wenn ersichtlich ist, dass der JetRacer pendelt, musst der `steering_gain_slider` reduziert werden
3. Schieberegler "Steering Bias": Ist zu nutzen, wenn der JetRacer nach rechts oder links verstimmt ist. Ziel sollte es sein, diesen Schieberegler so einstellen, dass der JetRacer der Linie oder der Spur in der Mitte folgt. Dies berücksichtigt sowohl die Motorfehler als auch die Kameraabweichungen.

   **Hinweis**: Anfangs sollten mit den oben genannten Schiebereglern mit geringerer Geschwindigkeit getestet werden, um ein gleichmäßiges Spurverfolgungsverhalten JetRacer-Straßenfolgeverhalten zu erreichen.

4. Schieberegler "Blockiert": Zeigt die Wahrscheinlichkeit an, mit der sich ein Hindernis vor dem JetRacer befindet, wobei das Kollisionsvermeidungsmodell verwendet wird.
5. Schieberegler Zeit für Stopp: Zur manuellen Einstellung der Zeit, die der JetRacer stehen bleiben soll, nachdem ein Objekt entfernt wurde

6. Schieberegler für Blockade: Zur manuellen Einstellung der Blockier-Schwelle, die den JetRacer nach der Erkennung eines Objekts anhält

[7]:
```
from ipywidgets import Layout, Button, Box, Label, HTML

box_layout = Layout(display='flex', align_items='stretch', border='dashed 1px',␣
 ↪width='650px')

##### JetRacer Control Items / Road Following sliders
speed_control_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01,␣
 ↪value=0.0, description='speed control',style = {'description_width':␣
 ↪'120px'})
slow_control_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01,␣
 ↪value=0.0, description='slowdown control',style = {'description_width':␣
 ↪'120px'})
steering_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01,␣
 ↪value=0.70, description='steering gain',style = {'description_width':␣
 ↪'120px'})
steering_dgain_slider = ipywidgets.FloatSlider(min=0.0, max=0.5, step=0.001,␣
 ↪value=0.0, description='steering kd',style = {'description_width': '120px'})
steering_bias_slider = ipywidgets.FloatSlider(min=-0.3, max=0.3, step=0.01,␣
 ↪value=0.0, description='steering bias',style = {'description_width':␣
 ↪'120px'})
#####

road_follow_box = ipywidgets.VBox([HTML(value = f"<b>{'JetRacer control items:
 ↪'}</b>"),speed_control_slider, slow_control_slider, steering_gain_slider,␣
 ↪steering_dgain_slider, steering_bias_slider],style = {'description_width':␣
 ↪'initial'})

##### Collision avoidance control items / Collision Avoidance sliders
slow_threshold= ipywidgets.FloatSlider(min=0, max=1.0, step=0.01, value=0.8,␣
 ↪description='slow threshold',style = {'description_width': '120px'})
blocked_threshold= ipywidgets.FloatSlider(min=0, max=1.0, step=0.01, value=0.8,␣
 ↪description='blocked threshold', style = {'description_width': '120px'})
stopduration_slider= ipywidgets.IntSlider(min=0, max=100, step=1, value=1, ␣
 ↪description='time for stop', style = {'description_width': '120px'})
#####

coll_avo_box= ipywidgets.VBox([HTML(value = f"<b>{'Collision avoidance control␣
 ↪items:'}</b>"), slow_threshold, blocked_threshold, stopduration_slider],␣
 ↪style = {'description_width': 'initial'})

#####Current Value for Collision avoidance
free_slider = ipywidgets.FloatProgress(min=0.0, max=1.0,␣
 ↪orientation='vertical',description='free', bar_style='success')
```

```
slow_slider = ipywidgets.FloatProgress(min=0.0, max=1.0,␣
 ↪orientation='vertical', description='slow',bar_style='warning')
blocked_slider = ipywidgets.FloatProgress(min=0.0, max=1.0,␣
 ↪orientation='vertical', description='blocked', bar_style='danger')
brake_level = ipywidgets.IntSlider(min=0, max=4, orientation='vertical',␣
 ↪description='brake-level', bar_style='waring')
current_speed = ipywidgets.Text(value=str(speed_control_slider.value),␣
 ↪description='Current speed:', style = {'description_width': '120px'},␣
 ↪disabled=True)
current_steering = ipywidgets.FloatSlider(min=-1.0, max=1.0,␣
 ↪description='Current steering:', style = {'description_width': '120px'},␣
 ↪layout=Layout(width='350px'))
#####

current_value_box = ipywidgets.HBox([free_slider, slow_slider, blocked_slider,␣
 ↪brake_level])

image = ipywidgets.VBox([HTML(value = f"<b>{'JetRacer view:'}</
 ↪b>"),image_widget], layout=Layout(width='224px'))
right = ipywidgets.VBox([HTML(value = f"<b>{'Collision prediction:'}</
 ↪b>"),current_value_box], style = {'description_width': 'initial'})

# Display all  Sliders and Control Items
display(ipywidgets.HBox([road_follow_box, coll_avo_box], layout=box_layout))
display(ipywidgets.HBox([image, right], layout=box_layout ))
display(ipywidgets.VBox([current_steering,current_speed], layout=box_layout ))

control_buttons = [
    Button(description='Start JetRacer', layout=Layout(width='200px'),␣
 ↪button_style='success'),
    Button(description='Stop JetRacer', layout=Layout(width='450px'),␣
 ↪button_style='danger'),
]

def on_button_clicked(btn):
    if btn.description == 'Start JetRacer':
        speed_control_slider.value = 0.18
        slow_control_slider.value = 0.16
        steering_gain_slider.value = 0.7
        with output:
            print("Start Button clicked.")
    elif btn.description == 'Stop JetRacer':
        speed_control_slider.value = 0
        slow_control_slider.value = 0
        steering_gain_slider.value = 0
        with output:
```

```
            print("Stopp Button clicked.")
control_buttons[0].on_click(on_button_clicked)
control_buttons[1].on_click(on_button_clicked)
Box(control_buttons)
```

```
HBox(children=(VBox(children=(HTML(value='<b>JetRacer Control Items:</b>'),␣
 ↪FloatSlider(value=0.0, description…
```

```
HBox(children=(VBox(children=(HTML(value='<b>JetRacer View:</b>'),␣
 ↪Image(value=b'\xff\xd8\xff\xe0\x00\x10JFIF\…
```

```
VBox(children=(FloatSlider(value=0.0, description='Current steering:',␣
 ↪layout=Layout(width='350px'), max=1.0, …
```

```
Box(children=(Button(button_style='success', description='Start JetRacer',␣
 ↪layout=Layout(width='200px'), style…
```

Erläuterung zur Geschwindigkeitsreglung
Das Modell sieht 3 Kategorien vor: 1. Umfeld frei - kein Fahrzeug in der Nähe 2. Fahrzeug im Umfeld erkannt 3. Umfeld blockiert

Entsprechend dieser 3 Kategorien erfolgt eine Auswahl des Bremslevel zur Geschwindigkeitsanpassung des JetRacers: 1. Level 0 - volle JetRacer Geschwindigkeit 0.75-1.00 - free; 2. Level 1 - geringe Geschwindigkeitsreduktion 0.00-0.75 - slow; 0.00-0.75 - free 3. Level 2 - mäßig Geschwindigkeitsreduktion 0.75-1.0 - slow 4. Level 3 - starke Geschwindigkeitsreduktion 0.00-0.75 slow & 0.00-0.75 - blocked 5. Level 4 - volle Geschwindigkeitsreduktion - Stillstand 0.75-1.00 - blocked

Als Nächstes wird eine Funktion erstellt, die immer dann aufgerufen wird, wenn sich der Wert der Kamera ändert. Die Funktion führt dabei folgende Schritte aus:

1. Vorverarbeitung des Kamerabildes
2. Ausführen der neuronalen Netzmodelle für Straßenverfolgung und Kollisionsvermeidung
3. Berechnung des ungefähren Lenkwinkels
4. Ansteuerung der Motoren
5. Prüfen, ob eine if-Anweisung, die es dem JetRacer ermöglicht, der Straße zu folgen und anzuhalten, wenn ein Hindernis erkannt wurde.
6. Berechne den ungefähren Lenkwert
7. Steuerung der Motoren mittels Proportional-/Differenzialsteuerung (PD)

[8]:
```
import math

angle = 0.0
angle_last = 0.0
count_stops = 0
go_on = 1
stop_time = 10 # The number of frames to remain stopped
x = 0.0
y = 0.0
speed_value = speed_control_slider.value

def execute(change):
```

```
    global angle, angle_last, blocked_slider, free_slider, slow_slider, robot,␣
↪count_stops, stop_time, go_on, x, y, blocked_threshold
    global speed_value, steer_gain, steer_dgain, steer_bias

    steer_gain = steering_gain_slider.value
    steer_dgain = steering_dgain_slider.value
    steer_bias = steering_bias_slider.value

    image_preproc = preprocess(change['new']).to(device)

    #Collision Avoidance model:

    prob_blocked = float(F.softmax(model_trt_collision(image_preproc), dim=1).
↪flatten()[0])
    prob_free = float(F.softmax(model_trt_collision(image_preproc), dim=1).
↪flatten()[1])
    prob_slow = float(F.softmax(model_trt_collision(image_preproc), dim=1).
↪flatten()[2])

    blocked_slider.value = prob_blocked
    free_slider.value = prob_free
    slow_slider.value = prob_slow

    stop_time=stopduration_slider.value

    if go_on == 1:

        # Brake-Level 0
        if prob_free > 0.75 and prob_slow < 0.25 and prob_blocked < 0.25:
            # prob_blocked > blocked_threshold.value:  threshold should be␣
↪above 0.5
            brake_level.value = 0
            go_on = 1
            count_stops = 0
            speed_value = speed_control_slider.value

        # Brake-Level 1
        elif 0.1 < prob_free < 0.75 and prob_slow < 0.75 and prob_blocked < 0.
↪25: # prob_slow > slow_threshold.value:
            brake_level.value = 1
            go_on = 1
            count_stops = 0
            speed_value = (speed_control_slider.value + slow_control_slider.
↪value) / 2

        # Brake-Level 2
```

```
        elif prob_free < 0.25 and prob_slow > 0.75 and prob_blocked < 0.25: #␣
↪prob_slow > slow_threshold.value:
            brake_level.value = 2
            go_on = 1
            count_stops = 0
            speed_value = slow_control_slider.value

        # Brake-Level 3
        elif prob_free < 0.25 and 0.1 < prob_slow < 0.75 and prob_blocked < 0.
↪75: # prob_slow > slow_threshold.value:
            brake_level.value = 3
            go_on = 1
            count_stops = 0
            speed_value = slow_control_slider.value*0.8
        # Brake-Level 4
        elif prob_free < 0.25 and prob_slow < 0.25 and prob_blocked > 0.75: #␣
↪prob_slow > slow_threshold.value:
            brake_level.value = 4
            speed_value = 0.0
            count_stops += 1
            go_on = 2
    else:
        count_stops += 1
        if count_stops < stop_time:
            x = 0.0 #set x steering to zero
            y = 0.0 #set y steering to zero
            speed_value = 0.0
        else:
            go_on = 1
            count_stops = 0
    xy = model_trt(image_preproc).detach().float().cpu().numpy().flatten()
    x = xy[0]
    y = (0.5 - xy[1]) / 2.0
    angle = math.atan2(x, y)
    pid = angle * steer_gain + (angle - angle_last) * steer_dgain
    steer_val = pid + steer_bias
    angle_last = angle

    current_speed.value = str(speed_value)

    robot.steering =  -steer_val

    robot.throttle = -speed_value

execute({'new': camera.value})
```

Somit wurde eine Funktion zur Ausführung des neuronalen Netzes erstellt, welche jedoch noch mit

der Kamera verbunden werden muss. Dies erfolgt über die `observe`-Funktion. > Warnung: Die nachfolgende Code-Zeile bewegt den JetRacer bzw. die Lenkung autonom und die Geschwindigkeitskontrolle wird über den Controller manuell gesteuert. Es ist sicherzustellen, dass der JetRacer sich in einem freien Raum befindet und auf der Spur/ Straße, auf der die Daten gesammelt wurden. Dabei ist die Spurverfolgung des neuronalen Netzes nur so gut wie die Daten, mit denen es trainiert wurde!

[9]:
```
camera.observe(execute, names='value')
```

Wenn der JetRacer angeschlossen ist, sollte er jetzt mit jedem neuen Kamerabild neue Befehle erzeugen. Soll der JetRacer gestoppt werden, kann der Callback durch Ausführen des folgenden Codes deaktiviert werden.

[ ]:
```
import time

camera.unobserve(execute, names='value')

time.sleep(0.1)  # add a small sleep to make sure frames have finished␣
↪processing
```

Die Kameraverbindung muss wieder ordnungsgemäß geschlossen werden, damit die Kamera in anderen Notebooks verwendet werden kann.

# Literaturverzeichnis


[1] J. Ritz, *Mobilitätswende - autonome Autos erobern unsere Strassen: Ressourcenverbrauch, Ökonomie und Sicherheit*. Wiesbaden: Springer, 2018, ISBN: 9783658209520.

[2] Umweltbundesamt, *Fahrleistungen, Verkehrsleistung und "Modal Split Umweltbundesamt*. Adresse: https://www.umweltbundesamt.de/daten/verkehr/fahrleistungen-verkehrsaufwand-modal-split#fahrleistung-im-personen-und-guterverkehr (besucht am 08. 09. 2022).

[3] Bundesamt, *6,6 % weniger Verkehrstote im Jahr 2019*, de. Adresse: https://www.destatis.de/DE/Presse/Pressemitteilungen/2020/02/PD20_061_46241.html (besucht am 28. 09. 2022).

[4] D. motorradonline, *Motorrad-Unfallstatistik 2019: Tötungsrisiko viermal so hoch*, de, Aug. 2020. Adresse: https://www.motorradonline.de/ratgeber/statistik-zu-motorradunfallen-2019-toetungsrisiko-vier-mal-so-hoch/ (besucht am 28. 09. 2022).

[5] E. Dönges und K. Naab, „Regelsysteme zur Fahrzeugführung und -Stabilisierung in der Automobiltechnik“, en, *auto*, Jg. 44, Nr. 5, S. 226–237, Mai 1996, ISSN: 2196-677X, 0178-2312. DOI: 10.1524/auto.1996.44.5.226. Adresse: https://www.degruyter.com/document/doi/10.1524/auto.1996.44.5.226/html (besucht am 10. 10. 2022).

[6] E. Donges, S. Hakuli, F. Lotz und C. Singer, *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort* (Handbuch Fahrerassistenzsysteme), ger, 3., überarb. u. erg. Aufl. 2015. Wiesbaden: Springer Vieweg, 2015, ISBN: 9783658057343.

[7] L. Eckstein, T. Dittmar, A. Zlocki und T. Woopen, „Automatisiertes Fahren — Potenziale, Herausforderungen und Lösungsansätze“, de, *ATZ - Automobiltechnische Zeitschrift*, Jg. 120, Nr. S3, S. 58–63, Aug. 2018, ISSN: 0001-2785, 2192-8800. DOI: 10.1007/s35148-018-0099-z. Adresse: https://link.springer.com/10.1007/s35148-018-0099-z (besucht am 03. 10. 2022).

[8] ADAC, *Dieses Auto fährt im Stau von selbst*, de-DE. Adresse: https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/autonomes-fahren/technik-vernetzung/autonomes-fahren-staupilot-s-klasse/ (besucht am 06. 10. 2022).

[9] h. heise heise, *Hochautomatisiertes Fahren: Level 3 bis 130 km/h zuerst bei Mercedes*, de. Adresse: https://www.heise.de/hintergrund/Hochautomatisiertes-Fahren-Level-3-bis-130-km-h-zuerst-bei-Mercedes-7162660.html (besucht am 06. 10. 2022).

[10] D. Brüggemann u. a., „Ansätze zur Verbesserung KI-basierter Systeme für das autonome Fahren“, de, in *Qualitätsmanagement in den 20er Jahren - Trends und Perspektiven*, B. Leyendecker, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, S. 100–119, ISBN: 9783662632420. DOI: 10.1007/978-3-662-63243-7_6. Adresse: https://link.springer.com/10.1007/978-3-662-63243-7_6 (besucht am 12. 10. 2022).

[11] K. Reif, Hrsg., *Fahrstabilisierungssysteme und Fahrerassistenzsysteme*, de. Wiesbaden: Vieweg+Teubner, 2010, ISBN: 9783834813145. DOI: 10.1007/978-3-8348-9717-6. Adresse: http://link.springer.com/10.1007/978-3-8348-9717-6 (besucht am 10. 10. 2022).

[12] Darms Michael, „Eine Basis-Systemarchitektur zur Sensordatenfusion von Umfeldsensoren für Fahrerassistenzsysteme“, Diss., Technischen Universität Darmstad, Apr. 2007. Adresse: https://www.fzd.tu-darmstadt.de/media/fachgebiet_fzd/previous_design/publikationen_3/2007/2007_darms_dissertation.pdf.

[13] ISO/TC 204, *ISO 15622:2018*, en, 2018. Adresse: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/15/71515.html (besucht am 11. 10. 2022).

[14] ISO/TC 204, *ISO 17361:2017*, en, 2017. Adresse: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/23/72349.html (besucht am 12. 10. 2022).

[15] ISO/TC 204, *ISO 11270:2014*, en, Intelligent transport systems — Lane keeping assistance systems (LKAS) — Performance requirements and test procedures, 2014. Adresse: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/03/50347.html (besucht am 12. 10. 2022).

[16] Volkswagen, *Travel Assist*, de, Dez. 2022. Adresse: https://www.volkswagen.de/idhub/content/experience-fragments/onehub_pkw/de/de/static/layers/vmc/iq_drive/travel-assist/master.html (besucht am 12. 10. 2022).

[17] *Mercedes-Benz Drive Pilot*, de-DE, Dez. 2022. Adresse: https://www.mercedes-benz.de/passengercars/technology-innovation/mercedes-benz-drive-pilot.html (besucht am 12. 10. 2022).

[18] Tesla, *Künstliche Intelligenz & Autopilot*, de-DE, 2022. Adresse: https://www.tesla.com/AI (besucht am 15. 10. 2022).

[19] *Image Classification on ImageNet*, en, 2022. Adresse: https://paperswithcode.com/sota/image-classification-on-imagenet (besucht am 15. 10. 2022).

[20] InterviewBit, *Deep Learning Vs Machine Learning: Difference Between Deep Learning and Machine Learning*, Juni 2022. Adresse: https://www.interviewbit.com/blog/deep-learning-vs-machine-learning/#:~:text=Deep%20learning%20models%20mostly%20deal%20with%20datasets%20having%20millions%20of%20data%20rows.&text=ML%20models%20take%20less%20time,because%20of%20massive%20data%20points.&text=Machine%20learning%20models%20are%20easy,interaction%20to%20make%20better%20predictions. (besucht am 24. 11. 2022).

[21] W. Ertel, *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung* (Computational Intelligence), de. Wiesbaden: Springer Fachmedien Wiesbaden, 2021, ISBN: 9783658320744. DOI: 10.1007/978-3-658-32075-1. Adresse: https://link.springer.com/10.1007/978-3-658-32075-1 (besucht am 15. 10. 2022).

[22] Jedox, *Zukunftstechnologien für BI und Controlling – Vorteile durch KI (2. Teil)*, de, Juli 2018. Adresse: https://www.jedox.com/de/blog/kunstliche-intelligenz-controlling-teil-2/ (besucht am 22. 10. 2022).

[23] Hussain Saleem, Khalid Bin Muhammad, Altaf Hussain Nizamani, S. Saleem und B. Jamshed, „DATA SCIENCE AND MACHINE LEARNING APPROACH TO IMPROVE E-COMMERCE SALES PERFORMANCE ON SOCIAL WEB“, *IAEME*, INTERNATIONAL JOURNAL OF ADVANCED RESEARCH IN ENGINEERING AND TECHNOLOGY (IJARET), Jg. 12, Nr. 4, S. 401–424, Apr. 2021. DOI: 10.34218/IJARET.12.4.2021.040. Adresse: https://iaeme.com/Home/issue/IJARET?Volume=12&Issue=4.

[24] C. Aichele und J. Herrmann, Hrsg., *Betriebswirtschaftliche KI-Anwendungen: digitale Geschäftsmodelle auf Basis Künstlicher Intelligenz*, ger. Wiesbaden [Heidelberg]: Springer Vieweg, 2021, ISBN: 9783658335311.

[25] L. Wuttke, *Praxisleitfaden für Künstliche Intelligenz in Marketing und Vertrieb: Beispiele, Konzepte und Anwendungsfälle*, de. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, ISBN: 9783658356255. DOI: 10.1007/978-3-658-35626-2. Adresse: https://link.springer.com/10.1007/978-3-658-35626-2 (besucht am 22. 10. 2022).

[26] M. Nolting, *Künstliche Intelligenz in der Automobilindustrie: mit KI und Daten vom Blechbieger zum Techgiganten* (Technik im Fokus), ger. Wiesbaden [Heidelberg]: Springer Vieweg, 2021, ISBN: 9783658315665.

[27] J. Steinwendner und R. Schwaiger, *Neuronale Netze programmieren mit Python* (Rheinwerk Computing), ger, 2., aktualisierte und überarbeitete Auflage, 1. korrigierter Nachdruck. Bonn: Rheinwerk Verlag, 2020, ISBN: 9783836274500.

[28] D. Frick u. a., Hrsg., *Data Science: Konzepte, Erfahrungen, Fallstudien und Praxis*, ger. Wiesbaden [Heidelberg]: Springer Vieweg, 2021, ISBN: 9783658334024.

[29] K. Kersting, C. Lampert und C. Rothkopf, Hrsg., *Wie Maschinen lernen: Künstliche Intelligenz verständlich erklärt*, de. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, ISBN: 9783658267629. DOI: 10.1007/978-3-658-26763-6. Adresse: http://link.springer.com/10.1007/978-3-658-26763-6 (besucht am 22. 10. 2022).

[30] Mathworks, *Convolutional Neural Network*, de. Adresse: https://de.mathworks.com/discovery/convolutional-neural-network-matlab.html (besucht am 22. 10. 2022).

[31] Nvidia, *Jetson Nano*, en-US, März 2019. Adresse: https://developer.nvidia.com/embedded/jetson-nano (besucht am 24. 10. 2022).

[32] Waveshare, *JetRacer Assembly Manual - Waveshare Wiki*, 2022. Adresse: https://www.waveshare.com/wiki/JetRacer_Assembly_Manual (besucht am 24. 10. 2022).

[33] Waveshare, *JetRacer AI Kit, AI Racing Robot Powered by Jetson Nano*, 2022. Adresse: https://www.waveshare.com/product/ai/boards-kits/jetson-nano/jetracer-ai-kit.htm (besucht am 24. 10. 2022).

[34] tensorflow, *Models*, original-date: 2016-02-05T01:15:20Z, Okt. 2022. Adresse: https://github.com/tensorflow/models/tree/master/research/slim (besucht am 25. 10. 2022).

[35] Waveshare, *Setup JetRacer AI Kit*, en, 2022. Adresse: https://www.waveshare.com/wiki/JetRacer_AI_Kit (besucht am 24. 10. 2022).

[36] G. Halfacree, *Raspberry Pi 4 auf dem Prüfstand*, de-DE, 2019. Adresse: https://www.raspberry-pi-geek.de/ausgaben/rpg/2019/10/raspberry-pi-4-auf-dem-pruefstand/ (besucht am 24. 10. 2022).

[37] J. Schmidt, C. Klüver und J. Klüver, *Programmierung naturanaloger Verfahren*, de. Wiesbaden: Vieweg+Teubner, 2010, ISBN: 9783834808226. DOI: 10.1007/978-3-8348-9666-7. Adresse: http://link.springer.com/10.1007/978-3-8348-9666-7 (besucht am 24. 10. 2022).

[38] Nvidia, *JetPack*, en-us, topic, 2022. Adresse: http://docs.nvidia.com/jetson/jetpack/introduction/index.html (besucht am 25. 10. 2022).

[39] PyTorch, *PyTorch*, 2022. Adresse: https://github.com/pytorch/pytorch (besucht am 02. 11. 2022).

[40] Waveshare, *Recovery Mode JETSON-NANO-DEV-KIT - Waveshare Wiki*, en. Adresse: https://www.waveshare.com/wiki/JETSON-NANO-DEV-KIT#System_Installation (besucht am 25. 10. 2022).

[41] X. Ou u. a., „Moving Object Detection Method via ResNet-18 With Encoder–Decoder Structure in Complex Scenes“, *IEEE Access*, Jg. 7, S. 108 152–108 160, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2931922. Adresse: https://ieeexplore.ieee.org/document/8781779/ (besucht am 03. 11. 2022).

[42] K. He, X. Zhang, S. Ren und J. Sun, „ResNet - Deep Residual Learning for Image Recognition“, 2015. DOI: 10.48550/ARXIV.1512.03385. Adresse: https://arxiv.org/abs/1512.03385 (besucht am 02. 11. 2022).

[43] F. Zhuang u. a., „A Comprehensive Survey on Transfer Learning“, *Proceedings of the IEEE*, Jg. 109, Nr. 1, S. 43–76, Jan. 2021, ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2020.3004555. Adresse: https://ieeexplore.ieee.org/document/9134370/ (besucht am 02. 11. 2022).

[44] S. Luber, *Was ist ein Directed Acyclic Graph (DAG)?*, de, 2022. Adresse: https://www.bigdata-insider.de/was-ist-ein-directed-acyclic-graph-dag-a-1075296/ (besucht am 19. 11. 2022).

[45] *torch.nn.functional — PyTorch 1.13 documentation*. Adresse: https://pytorch.org/docs/stable/nn.functional.html (besucht am 18. 11. 2022).

[46] *NVIDIA TensorRT*, en-US, Apr. 2016. Adresse: https://developer.nvidia.com/tensorrt (besucht am 25. 11. 2022).

[47] *torch2trt*, original-date: 2019-04-27T15:30:56Z, Nov. 2022. Adresse: https://github.com/NVIDIA-AI-IOT/torch2trt (besucht am 25. 11. 2022).

[48] *Accelerating Inference Up to 6x Faster in PyTorch with Torch-TensorRT*, en-US, Dez. 2021. Adresse: https://developer.nvidia.com/blog/accelerating-inference-up-to-6x-faster-in-pytorch-with-torch-tensorrt/ (besucht am 25. 11. 2022).

[49] *PyTorch for Jetson - Jetson & Embedded Systems / Jetson Nano*, de, März 2019. Adresse: https://forums.developer.nvidia.com/t/pytorch-for-jetson/72048 (besucht am 28. 11. 2022).

[50] T. maintainers und contributors, *TorchVision: PyTorch's Computer Vision library*, original-date: 2016-11-09T23:11:43Z, Nov. 2016. Adresse: https://github.com/pytorch/vision (besucht am 28. 11. 2022).

[51] Q-engineering, *Install PyTorch on Jetson Nano - Q-engineering*, en-GB. Adresse: https://qengineering.eu/install-pytorch-on-jetson-nano.html (besucht am 29. 11. 2022).

# Eidesstattliche Erklärung

Hiermit versichere ich – Christian Schreiber – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

| Mittweida, 02. Februar 2023 | |
|---|---|
| Ort, Datum | Christian Schreiber, Dipl.-Ing. (FH) |