Spurerkennung aus Kameradaten mittels maschinellem Lernen

Sebastian Gedigk
Department Informatik, Hochschule für angewandte Wissenschaften Hamburg
sebastian.gedigk@haw-hamburg.de

Zusammenfassung

Das selbständige erkennen von Kurven und Spurverläufen eines Fahrzeugs mittels Bilderkennung durch maschinelles Lernen wird in autonomen Fahrzeugen eingesetzt (Fraunhofer-Institut, o. J.). Ziel dieser Arbeit ist es, mit wenigen Bildern ein Neuronales Faltungsnetz zu trainieren, sodass diese Aufgaben bewältigt werden können. Das Neuronale Netz ist ein ResNet34. Dazu wurde ein Fahrzeug im Maßstab 1:10 verwendet, ein Nvidia Jetson Nano und eine Kamera. Der Jetson kann durch seine integrierte Grafische-Prozessoreinheit komplexe „Machine-Learning“ Berechnungen in kurzer Zeit durchführen. Die gute Rechenleistung des Jetson macht eine schnelle Bildverarbeitung und Inferenz des ResNet möglich. Es wurden circa 1000 Bilder benötigt, damit das Fahrzeug selbstständig Kurven und Streckenverläufe auf einer vorgegebene Strecke erkennen und darauf reagieren kann. Das Fahrzeug hat sich erfolgreich durch einen Kreisverkehr im Uhrzeigersinn, Kreisverkehr entgegen dem Uhrzeigersinn und einer Strecke aus mehreren Geraden, Kurven und einem Rondell manövriert. Die größten Störfaktoren, für das erfolgreiche absolvieren dieser Tests, waren Lichtreflexionen auf dem Boden der Fahrbahn. Diese konnten dazu führen, dass das Fahrzeug sich nicht auf der Strecke halten konnte.

Keywords: Nvidia Jetson; Tamiya; Spurerkennung; Faltungsnetz

1 Einleitung

1.1 Motivation

Fahrzeuge besitzen heutzutage eine Vielzahl an Assistenzsystemen. Sie verfügen zum Beispiel über Spurhalteassistenten, Abstandshaltesysteme und Abbiegeassistenten. Auf dem Weg ein Fahrzeug autonom fahren zu lassen sind diese Systeme essentiell. Damit ein Fahrzeug selbstständig Kurven erkennen und darauf reagieren kann, muss es mit Hilfe von Sensoren wie Kameras, Lidar und Radar in der Lage sein, die Umgebung zu klassifizieren. Diese Klassifizierung findet mit maschinellem Lernen statt.

1.2 Aufgabe

Das Ziel dieses Projekts ist es, ein fern steuerbares Auto im Maßstab 1:10 selbstständig Kurven und Fahrbahnen erkennen zu lassen und darauf zu reagieren. Damit dies möglich ist benötigt das Fahrzeug als zusätzliche Hardware einen Computer und eine Kamera. Der Kern dieser Funktionalität ist ein neuronales Netz, welches mit Hilfe von Trainingsdaten in der Lage ist Fahrbahnverläufe zu erkennen und darauf mit einem angepassten Lenkeinschlag zu reagieren. Die Grundlage dieses Projekts ist das Jetsoncar von Nvidia (Yato, 2019).

1.3 Aufbau

Das Projekt ist in vier Phasen aufgebaut. Die erste Phase dient dazu das Fahrzeug mit der benötigten Hardware und Software zur Bildverarbeitung und Manövrierfähigkeit auszustatten. In der zweiten Phase wird ein neuronales Netz gewählt und implementiert um die Fahrbahn zu erkennen und einen Lenkeinschlag zu berechnen. Die dritte Phase wird genutzt um Trainingsdaten in Form von Bildern zu sammeln und anschließend das Netz zu trainieren. In der vierten und letzten Phase wird das trainierte Netz getestet. Dazu wird das Fahrzeug auf eine markierte Strecke gesetzt um diese Strecke entlang zu fahren. Wenn das Fahrzeug die Strecke ausreichend gut erkennt, ist die Phase vier und das Projekt abgeschlossen. Bei nicht ausreichend guter Erkennung der Strecke werden weitere Testdaten aufgenommen und das Netz neu trainiert bis das Ergebnis zufriedenstellend ist.

2 Projekt Ablauf

2.1 Phase 1: Hardware & Software konfigurieren

In diesem Projekt wird folgende Hardware benutzt:

- Nvidia Jetson Nano 4Gb
- Tamiya TT-02 Bausatz
- RC-Reflex Pro 3
- Carson Reflex Wheel Start
- RC Servo Multiplexer
- PWM Servotreiber 16 Kanal
- WiFi Chipsatz & WiFi Antennen
- Powerbank 10.000 mAh
- Weitwinkel Kameramodul für Jetson Nano

Damit das Fahrzeug mit den restlichen Komponenten verbunden werden kann, benötigt man eine Plattform. Diese Plattform wird im 3D-Druck verfahren hergestellt. Die 3D gedruckte Plattform ist so angepasst, dass die einzelnen Komponenten miteinander verschraubt werden können.

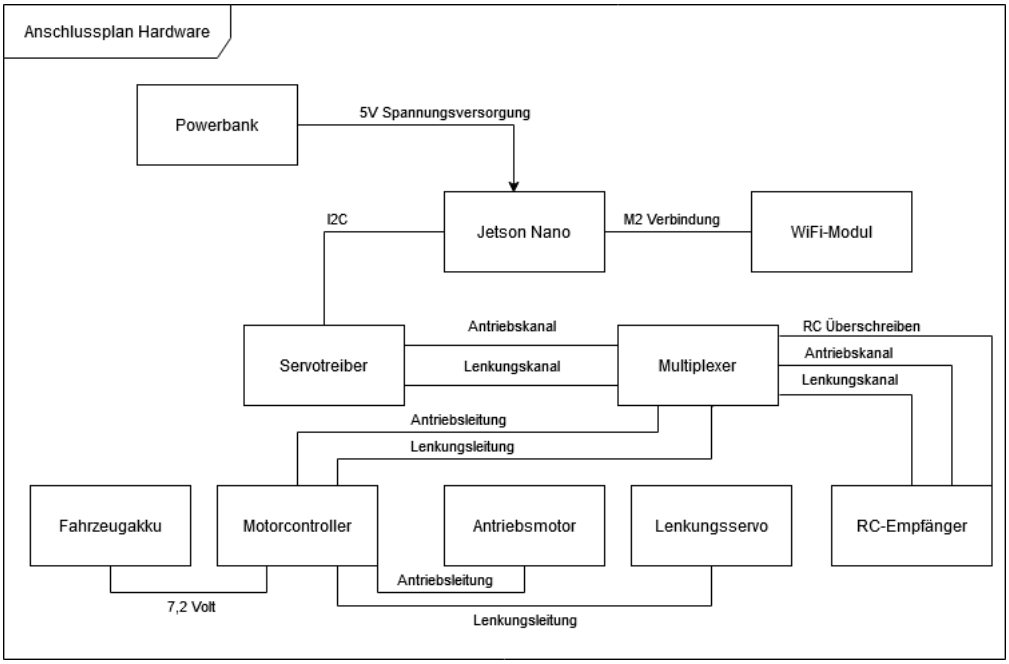


Abbildung 1: Anschlussplan

In Abbildung 1 ist der Anschlussplan der Elektronik dargestellt. Der Jetson Nano kann mit Hilfe des Servotreibers und des Multiplexers das Fahrzeug steuern. An der Fernbedienung wird die Fahrzeugsteuerung manuell für den Jetson freigegeben. Das ist eine Sicherheitsmaßnahme, damit bei einer Fehlfunktion schnellstmöglich die Steuerung des Fahrzeuges übernommen werden kann.

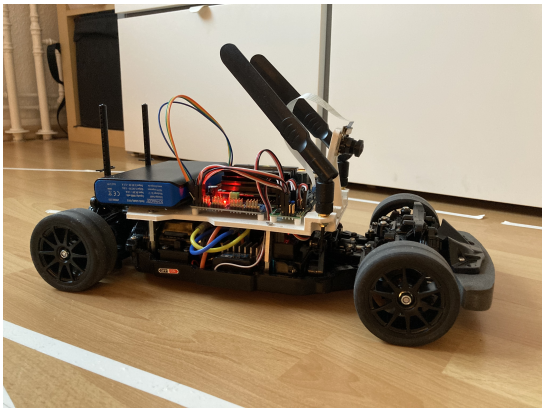
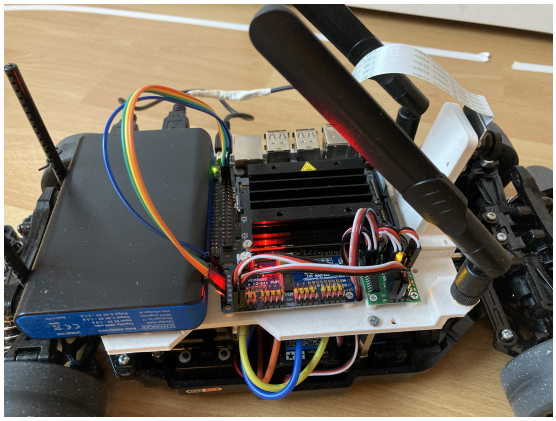
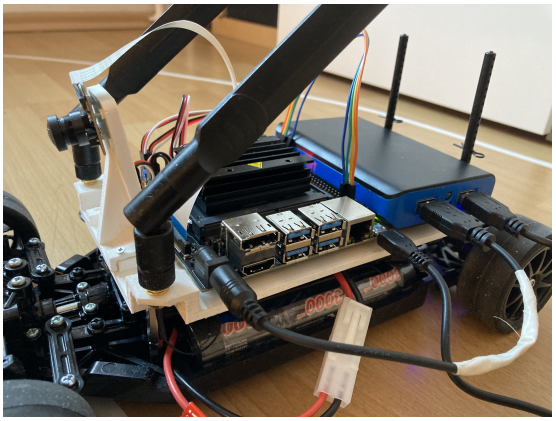


Abbildung 2: Komplettansicht des Fahrzeugs

Die Abbildung 2 zeigt das zusammengesetzte Fahrzeug mit angeschlossener Powerbank. Der Jetson wird dem eigenen WiFi hinzugefügt, damit er über das Netzwerk konfiguriert werden kann. Die Leistungsaufnahme des Jetson ist im Werkszustand 10 Watt. Diese Leistung ist für die Powerbank zu hoch. Deshalb wird der Jetson auf eine Leistungsaufnahme von 5 Watt um konfiguriert, wenn eine Powerbank ihn mit Strom versorgt.



(a) Nahaufnahme Fahrzeugseite rechts



(b) Nahaufnahme Fahrzeugseite links

Abbildung 3: Nahaufnahmen des Jetson-Car

Auf der Abbildung 3a ist die rechte Seite des Fahrzeugs zu sehen. Im Detail sieht man dort die Verkabelung des I2C Servotreibers und des Rc Multiplexers.
Damit die Powerbank eine hohe Leistung abgeben kann muss das Stromversorgungskabel sehr kurz sein. So können trotz des geringen Leiterquerschnitts relativ verlustfrei die 5 Watt Leistung an den Jetson abgegeben werden. Das gekürzte Stromkabel ist in Abbildung 3b zu sehen.
Die Software des Jetson ist vorkonfiguriert von Nvidia. Nach der Installation des Betriebssystems auf der SD-Karte des Jetson werden folgende drei weitere Pakete installiert:

- Jetcam zur Ansteuerung der Kamera.
- Torch2trt damit das Torch-Model in ein TensorRT-Model umgewandelt werden kann.
- Jetracer damit der Jetson das Auto steuern kann über I2C.

Der Zugriff auf den Jetson erfolgt über einen beliebigen Browser von einem Computer im Netzwerk. In diesem Projekt mit der zugewiesenen IP-Adresse 192.168.1.150. Es ist Jupyter Notebook auf dem Jetson vorinstalliert. Hierüber lässt sich der Jetson verwalten. Man kann direkt Jupyter Notebooks anlegen und damit programmieren oder über eine Terminal-Konsole auf den Jetson zugreifen.

2.2 Phase 2: Neuronales Netz wählen

Das Fahrzeug soll am Ende des Projekts, angepasst an den Verlauf einer markierten Strecke auf dem Boden, selbständig lenken. Damit das realisiert werden kann wird ein neuronales Netz genutzt. Es gibt eine Vielzahl an Möglichkeiten welches Netz man einsetzen kann. In diesem Projekt wird ein ResNet34 (He, Zhang, Ren & Sun, 2015)(Residual Network) genutzt. Das ResNet gehört zu den tieferen Netzen im Bereich des maschinellem Lernens. Es ist möglich mit einer geringen Anzahl an Bildern und Trainingsdaten eine hohe Genauigkeit in der Streckenerkennung zu erreichen. Das ResNet wird als fertig trainiertes Netz implementiert, das heißt es wird mit bereits gesetzten Gewichten und Bias Werten gestartet. Dieses Netz wurde gewählt, da es es sich mit der Anzahl der Schichten und der dafür hohen Genauigkeit gut auf dem Jetson trainieren

und berechnen lässt. Den Jetson zeichnet seine integrierte GPU aus. Diese Ausstattung lässt ihn viel schneller die Inferenz bei einem tiefen Netz durchführen als auf einem Raspberry Pi (NVIDIA, o. J.). Mit dieser GPU ist es dem Jetson möglich eigenständig komplexere Machine Learning Berechnungen durch zu führen.

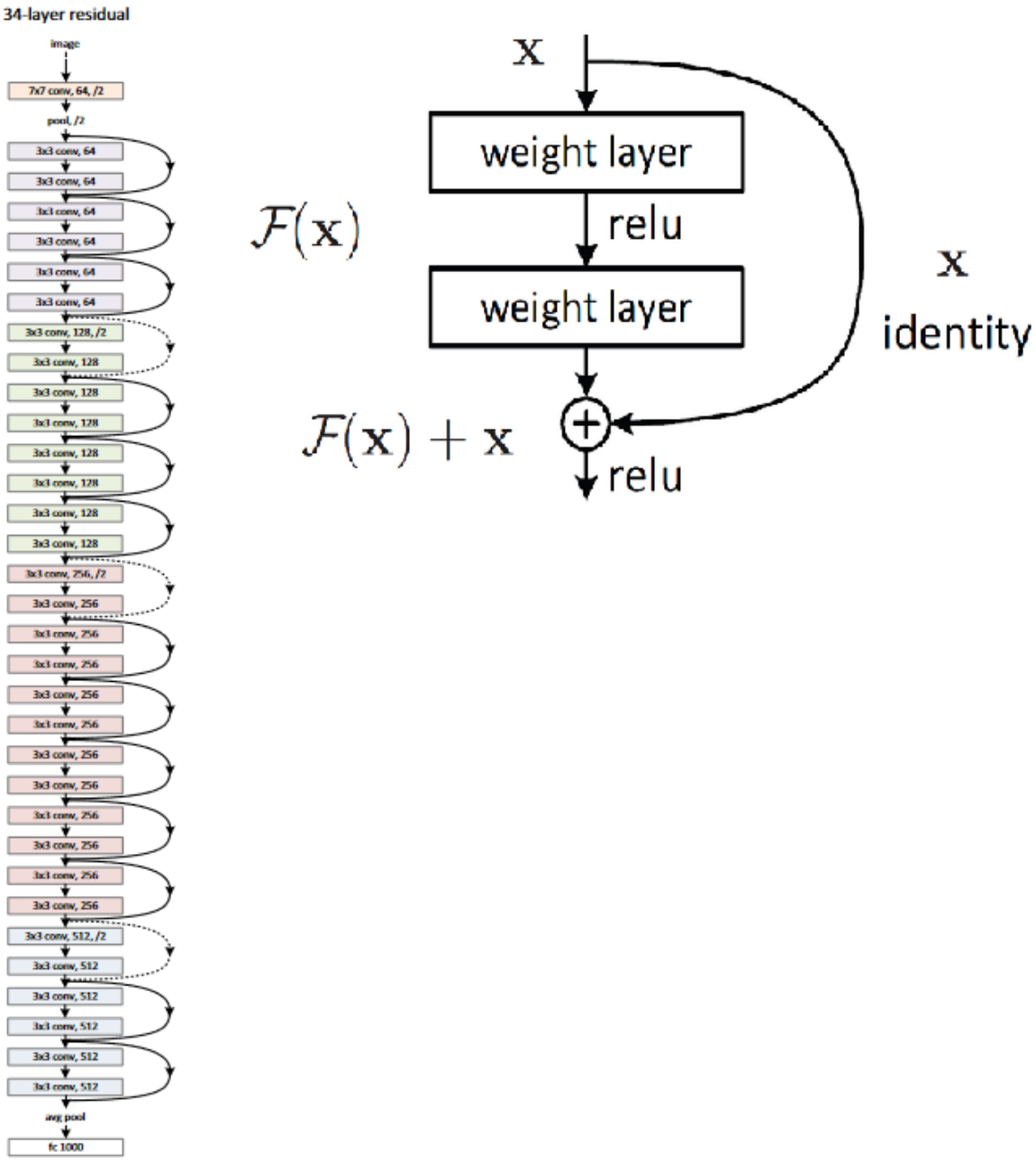


Abbildung 4: Aufbau des ResNet34 aus Fig.2 und Fig. 3 von Kaiming He (He et al., 2015)

Der Aufbau des ResNet ist in Abbildung 4 dargestellt. Ein Eingangsbild wird durch 34 Faltungsschichten gefaltet. Am Ende wird das Bild in eine von 1000 Klassen eingeteilt. Der Wert jeder Klasse ist im Bereich von -1 bis 1 verteilt. Dieser Wert wird dann mit in die Berechnung des Lenkwinkels fließen. Die Residual Netze nutzen die Identi-

tätsabbildungen von vorherigen Berechnungen und bilden damit “Shortcuts“ zwischen einer oder mehreren Schichten im Netz. Bei tiefen Netzen tritt im allgemeinen schnell das “Vanishing Gradient“-Problem auf. “Je tiefer das Netz wird, desto mehr partielle Ableitungen der Aktivierungsfunktion, d. h. Faktoren müssen miteinander multipliziert werden. Sind diese während des Trainings kleiner Eins, dann multiplizieren sich mehrere Faktoren zu einer Zahl, die schnell gegen Null strebt, weswegen die Gewichtsänderungen in tiefen Schichten deutlich langsamer sind als die in höheren. “(Wick, 2017). Um das zu umgehen nutzt ResNet die Identität zwischen den Schichten und ReLu als Aktivierungsfunktion.

2.3 Phase 3: Trainingsdaten sammeln & Netz trainieren

Die Trainingsdaten für das neuronale Netz werden manuell gesammelt. Dazu wird ein live Bild vom Fahrzeug an den Rechner übertragen, von dem aus auf das Fahrzeug zugegriffen wird. Das Bild hat die Auflösung 224x224 Pixel. Die Größe ist ausreichend, um alle nötigen Informationen aus der Strecke zu entnehmen. Außerdem lässt sich mit einer geringeren Auflösung schneller ein neuronales Netz trainieren.

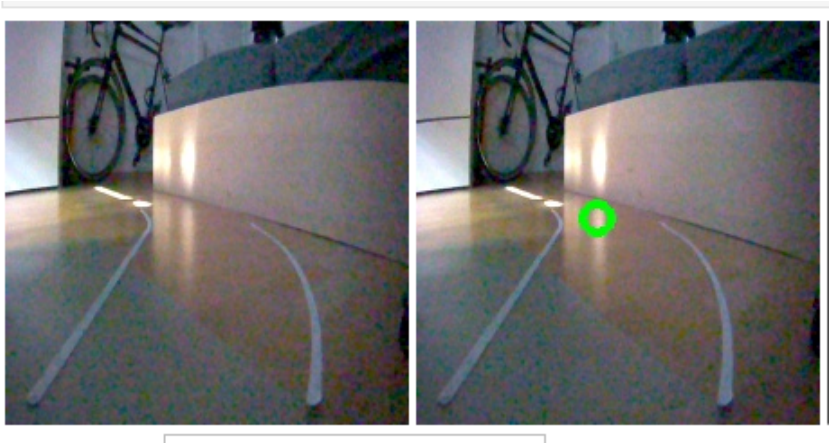
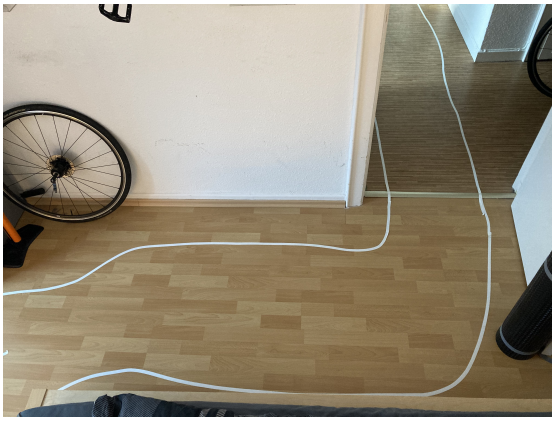


Abbildung 5: Ansicht Trainingsdaten sammeln

In der Abbildung 5 ist zu sehen, wie ein Livebild aussieht. Damit das Bild klassifiziert werden kann und abgespeichert wird, muss der Nutzer auf dem Bild auf die Stelle klicken, wo das Fahrzeug am besten entlang fahren soll. Die Position ist in etwa mit einer Ideallinie aus dem Rennsport zu vergleichen. Es spiegelt die Stelle wieder, an der das Fahrzeug entlang fahren soll, damit es die Strecke bestmöglich absolvieren kann. Die Position (X & Y Wert), auf die geklickt wurde wird dann im Dateinamen mit abgespeichert. So ist das Bild direkt klassifiziert für das Training. Damit das Netz eine ausreichende Anzahl an Trainingsdaten erhält, wurden ca. 1000 Bilder gemacht während das Fahrzeug auf dem Kurs bewegt wurde. Aufgrund der aktuellen Corona-Pandemie ist es nicht möglich gewesen, auf dem Hochschuleigenem Kurs (im Maßstab 1:10) das Netz zu trainieren. Deshalb wurde meine Wohnung, so gut es möglich war, als Versuchsstrecke umgebaut.

(a) Kurs Teil 1



(b) Kurs Teil 2

(c) Kurs Teil 3

(d) Kurs Teil 4

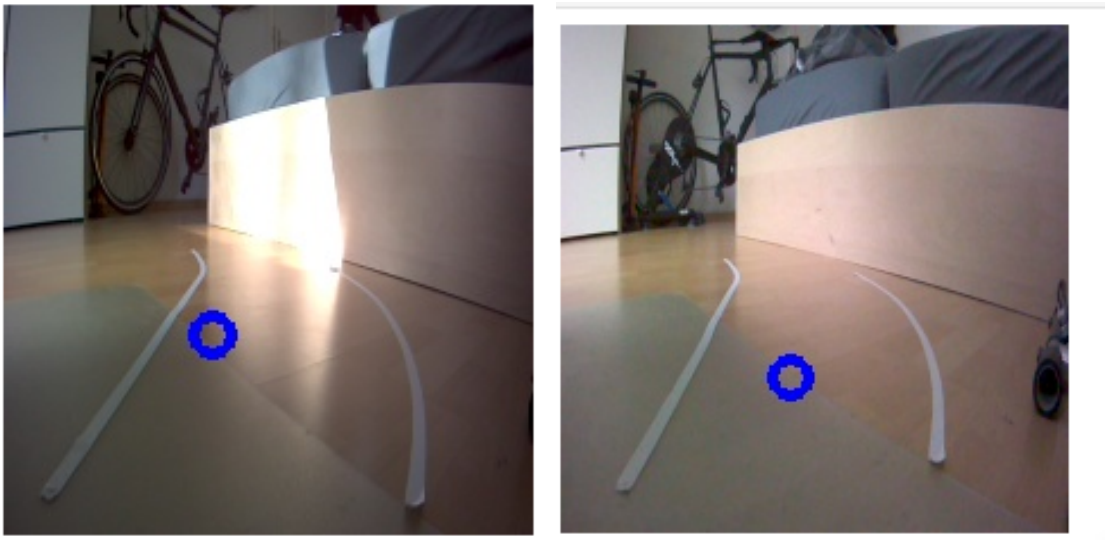
Abbildung 6: Darstellung des eigenen Kurses

In den Abbildungen 6a-d ist der Kurs abgebildet auf dem das Fahrzeug gefahren ist um das Netz zu trainieren. Der Kurs beinhaltet zwei Links- und Rechtskurven sowie ein Rondell mit sehr engem Radius (bedingt durch die Größe des Zimmers). Die Strecke wurde für das Training mehrmals abgefahren. Dazu wurde die Strecke aus verschiedenen Richtungen befahren und mehrere Runden im Rondell gedreht. Die Lichtverhältnisse waren während des Aufnehmens der Trainingsdaten relativ hell und es gab ein paar Lichtreflexionen am Boden auf dem Laminat. Nachdem genügend Bilder aufgenommen wurden, begann das Training der Daten.
Das Netz wurde für das Training der Daten wie folgt aufgebaut:

- Neuronales Faltungsnetz ResNet 34
- Batch-Size = 8
- Optimizer = Adam
- Aktivierungsfunktion ReLu
- Anz. Bilder = 1038
- Epochen = 15
- Anzahl Parameter: 21.285.698

2.4 Phase 4: Netz evaluieren & Fahrzeug auf der Strecke testen

Das trainierte Netz wird vor der Inbetriebnahme mit dem Lenkmotor getestet. Dazu wurde das Livebild mit dem Netz verknüpft und zeigt mit Hilfe eines blauen Kreises an, wohin das Fahrzeug lenken würde. Bei diesem Test ist aufgefallen, dass die Lichtverhältnisse eine wichtige Rolle spielen. Wenn es eine starke Lichtreflexion gibt wie in Abbildung 7a zu sehen, beeinflusst dieses das Netz wie weit eingelenkt wird. Diese Reflexionen stören das Netz den korrekten Lenkwinkel zu erkennen, da die Reflexion vermutlich als Streckenbegrenzung wahrgenommen wird. In Abbildung 7b sieht man, dass der Kreis bei der gleichen Position des Fahrzeugs ohne Lichtreflexion an anderer Stelle auf der Strecke gesetzt wurde.



(a) Lenkrichtung mit Lichtreflexion (b) Lenkrichtung ohne Lichtreflexion

Abbildung 7: Anwendung des Netzes auf das Live Bild der Kamera

Die Evaluierung des Netzes wurde auf der Strecke an bestimmten kritischen Stellen wie dem Kreisverkehr (Rondell) und an den Kuren ausprobiert. Diese Stellen wurden vom Netz korrekt erkannt. Zum Abschluss der Phase 4 wurde das Netz mit der Motorik des Fahrzeugs kombiniert und auf die Strecke gesetzt. Die Teststrecke wurde in 3 Teile aufgeteilt.

- Teil 1 Kreisverkehr im Uhrzeigersinn
- Teil 2 Kreisverkehr entgegen dem Uhrzeigersinn
- Teil 3 Komplette Strecke wie in Abbildung 6 dargestellt

Alle drei Teststrecken wurden vom Fahrzeug erkannt und konnten gut verfolgt werden. Damit das Fahrzeug nicht zu schnell wird beim befahren der Teststrecken wurde die geringste Geschwindigkeit genutzt und in Intervallen beschleunigt. So war es möglich ein Auto im Maßstab 1:10 in der Wohnung zu testen.

Quellcode 1: Code zum Anwenden des Konvertierten Modells auf dem Jetson und dem Fahrzeug

```
1  from utils import preprocess
2  import numpy as np
3  from jetcam.csi_camera import CSICamera
4  from jetracer.nvidia_racecar import NvidiaRacecar
5  import torch
6  from torch2trt import TRTModule
7
8  model_trt = TRTModule()
9  model_trt.load_state_dict(torch.load('road_following_model_trt.pth'))
10
11 car = NvidiaRacecar()
12
13 camera = CSICamera(width=224, height=224, capture_fps=65)
14
15
16 STEERING_GAIN = -1
17 STEERING_BIAS = 0.20
18 car.throttle = -0.2
19 car.throttle_gain = 0.15
20 i = 1
21 while True:
22     image = camera.read()
23     image = preprocess(image).half()
24     output = model_trt(image).detach().cpu().numpy().flatten()
25     x = float(output[0])
26
27     car.steering = x * STEERING_GAIN + STEERING_BIAS
28     if(i%2 != 0):
29         car.throttle = -0.2
30     else:
31         car.throttle = -0.0
32     if(i%100 ==0):
33         print(car.steering)
34     i+=1
```

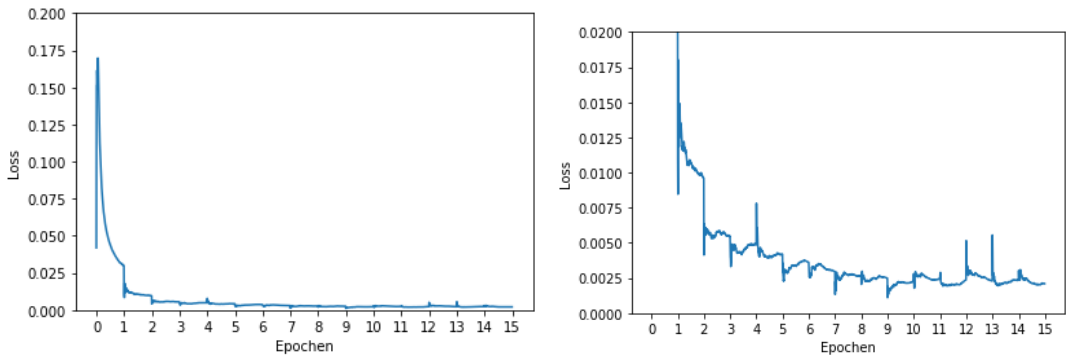
Damit das trainierte Modell mit einer hohen Bildrate vom Jetson verarbeitet werden kann wird es in ein TensoRt Modell umgewandelt. TensoRt ist optmiert für Inferenz bei tiefen Neuralen Netzen und ermöglicht eine schnelle Bildverarbeitung. Das angepasste Netz kann dann genutzt werden um das Fahrzeug zu lenken. In Quellcode 1 ist ein Ausschnitt zu sehen, der das TensorRt Modell lädt, die Kamera aktiviert und ein Fahrzeugobjekt erstellt. Ab Zeile 16 werden die Gewichte für die Lenkung und Beschleunigung definiert. In der While-Schleife ab Zeile 21 wird folgendes Prozedere ausgeführt:

- Aktuelles Bild einlesen
- Bild vorverarbeiten
- Inferenz des Modells mit dem aktuellen Bild
- Berechnete Position auslesen und mit den Gewichten und dem Bias der Lenkung den Lenkwinkel berechnen
- In Intervallen beschleunigen und alle 100 Bilder den Lenkwinkel ausgeben.

Wichtig beim Ausführen diese Codes ist es, immer das Fahrzeug zu beobachten, damit es bei einem Fehlverhalten direkt manuell gesteuert werden kann.

3 Auswertung

Die Strecken wurden vom ResNet gut erkannt. Das Fahrzeug ist in Teil 1 (dem Kreisverkehr im Uhrzeigersinn) ohne überschreiten der Markierungen dem Streckenverlauf gefolgt.
Teil 2 (Kreisverkehr entgegen dem Uhrzeigersinn) wurde fast genau so gut erkannt wie Teil 1 es gab jedoch ab und an Markierungsüberschreitungen im inneren des Kreises. Dies ist vermutlich auf die Bauart der Strecke zurück zu führen, denn die Kreise innen und außen wurden per Hand mit dem Isolierband gezogen und sind nicht exakt rund. Daher sind die Kurvenwinkel anders erkannt worden als in Teil 1.
Der Teil 3 wurde vom ResNet ausreichend gut erkannt, um den Kurs in 9 von 10 Fällen durchgehend ohne Eingreifen fahren zu können. Es kam jedoch vor, dass das Fahrzeug die Streckenbegrenzungen touchiert hat bzw. Wände und Schränke berührt wurden. Diese Berührungen waren jedoch nicht schwerwiegend, sodass das Fahrzeug selbständig weiter fahren und die Strecke beenden konnte. Der eine Fall von 10 führte dazu, dass das Fahrzeug in einer Kurve die Strecke komplett verließ und nicht den Weg zurück gefunden hat. Die meisten Ungenauigkeiten beim Erkennen der Strecke sind auf die Lichtverhältnisse zurück zu führen. Beim Erstellen der Trainingsdaten gab es einige Bilder auf denen starke Reflexionen des Sonnenlichts zu sehen waren. Diese Reflexionen sind mit Absicht in das Aufnehmen der Trainingsdaten eingegangen, damit das Netz diese Lichtverhältnisse bereits kennt. Es sind wahrscheinlich zu wenig Bilder mit Reflexionen aufgenommen worden, denn beim Anwenden des Netzes wurden Teile der Strecke mit Reflexionen ab und an nicht so gut erkannt wie ohne Reflexionen.
Das ResNet berechnet einen Wert zwischen -1 und 1. Dieser Wert wird mit dem Zugewinn-Wert (Gain) der Lenkung multipliziert und mit dem Bias-Wert addiert.



(a) Loss Komplett über 15 Epochen (b) Loss fokussiert auf Werte unter 0.01%

Abbildung 8: Loss des ResNet34 beim trainieren mit 15 Epochen

Das Netz hat beim Trainieren einen geringen Loss-Wert erzielt. Das ist in Abbildung 8a und 8b zu erkennen. Das bedeutet, dass die Vorhersagen des Netzes sehr gering von der Wirklichkeit abweichen. Das Trainieren eines solchen ResNet hat auf dem Jetson etwa 30-40 Minuten gedauert. Diese Zeit kann nur durch das Berechnen auf der Grafikeinheit erreicht werden. Ohne GPU würden die Berechnungen ein vielfaches länger dauern.

4 Fazit

Dieses Projekt ist erfolgreich durchgeführt worden. Das Ziel dieses Projekts wurde erfüllt, denn das Fahrzeug war in der Lage selbstständig durch eine vorgegebene Strecke zu manövrieren.
Es konnte mit circa 1000 Bildern ein gutes Ergebnis erzielt werden. Wenn man die Anzahl der Bilder weiter erhöht, wird das Netz voraussichtlich sehr an Genauigkeit gewinnen. Dabei ist zu beachten, dass mehr unterschiedliche Lichtverhältnisse bei den Trainingsdaten genutzt werden müssen zum Beispiel dunkle Fotos mit partiellem Licht als Laternen Ersatz. Die Lichtverhältnisse können aktuell beim Abfahren der Strecke dazu führen, dass das Fahrzeug sich fest fährt. Wenn das geschieht, könnte man in einem folgenden Projekt in Erwägung ziehen, dass das Fahrzeug bei nicht erkennen der Strecke zurücksetzt bis es wieder eine Strecke erkennt. Es ist von hoher Wichtigkeit, dass die genutzte Kamera ein breites Sichtfeld aufweist, damit aus jedem beliebigen Winkel, auf der Strecke, die Fahrbahnmarkierungen gut erkannt werden können.
Des weiteren könnte man in einem späteren Projekt versuchen die Geschwindigkeit ebenfalls vom ResNet anpassen zulassen. Dazu wird ein weiteres Modell für die Geschwindigkeitssteuerung oder eine neue Klassifizierung dem vorhandenen Modell hinzugefügt.
Nvidia hat mit dem Jetson ein leistungsfähiges Stück Hardware entwickelt, welches in der Lage ist komplexe Berechnungen ohne weitere Hardware durchzuführen.
Aufgrund des geringen Platzes in meiner Wohnung und der nicht mehr vorhandenen Möglichkeit die Teststrecke der Hochschule zu nutzen, wäre es besser gewesen, ein Fahrzeug in einem etwas kleinerem Maßstab zu nehmen, zum Beispiel im Maßstab 1:18 oder 1:24. Das erlaubt einem in der eigenen Wohnung eine Vielzahl von Teststrecken mit unterschiedlichsten Formen aufzubauen. So kann das Netz intensiver getestet werden.
Man könnte ebenfalls anstreben, mittels Unity oder anderer Software, das Netz auf einer virtuellen Straße zu trainieren, bevor es dann in der Realität eingesetzt wird.

Literatur

Fraunhofer-Institut. (o. J.). *Künstliche intelligenz (ki) und maschinelles lernen.* Web. Zugriff auf `https://www.iks.fraunhofer.de/de/themen/kuenstliche-intelligenz.html`

He, K., Zhang, X., Ren, S. & Sun, J. (2015). *Deep residual learning for image recognition.*

NVIDIA. (o. J.). *Jetson nano: Deep learning inference benchmarks.* Web. Zugriff auf `https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks`

Wick, C. (2017). *Deep learning.* Web. Springer-Verlag Berlin Heidelberg.

Yato, C. (2019, Juni). *Nvidia-ai-iot.* Web. Zugriff auf `https://github.com/NVIDIA-AI-IOT/jetracer`