ins
INSTITUTE FOR
NETWORKED SOLUTIONS

OST
Ostschweizer
Fachhochschule

JetBot

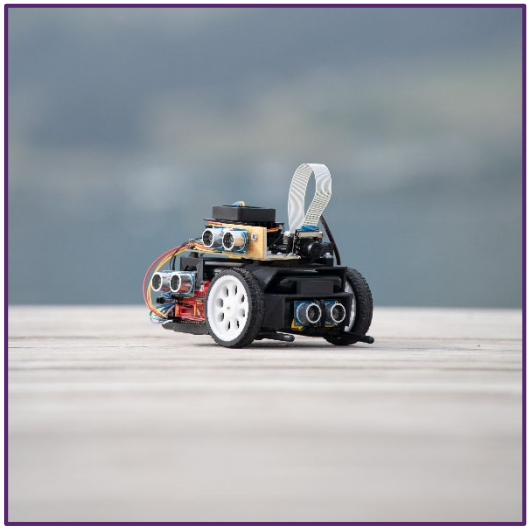
Präsentationsumgebung für angewandte Machine Learning Probleme

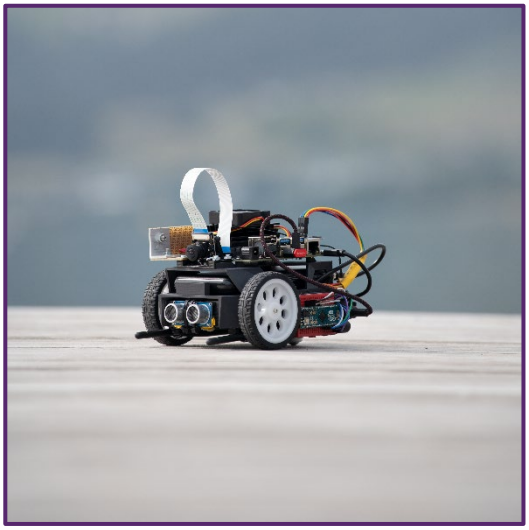
Bachelorarbeit

Studiengang Informatik

OST – Ostschweizer Fachhochschule

Campus Rapperswil-Jona





Frühjahrssemester 2021

Autoren:	Benjamin Peter, Yanick Rek, Aaron Studer
Betreuer:	Prof. Dr.-Ing. Andreas Rinkel, Marc Sommerhalder
Projektpartner:	INS, Institute for Networked Solutions
Experte:	Knut Schmahl
Gegenleser:	Prof. Stefan F. Keller

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

1. Abstract

Diese Arbeit, welche auf der Studienarbeit «JetBot Autonomes und fortlaufendes maschinelles Lernen» basiert, hat zum Ziel, neu angehenden Studierenden, die noch am Beginn ihrer Ausbildung stehen, Einsatzfelder im Bereich Künstliche Intelligenz näher zu bringen. Um diese möglichen Einsatzfelder interaktiv und ansprechend darzustellen, wird ein Roboter verwendet, der auf einem Nvidia Jetson Nano Developer Kit basiert und durch weitere Komponenten wie drei Ultraschallsensoren und einem Mikrocontroller ergänzt wurde. Das Fahrgestell des Roboters basiert auf einem 3D Modell, das für die zu lösenden Anwendungen optimiert wurde.

Die folgenden Probleme können von dem Roboter autonom gelöst werden:

- Erkennen eines Objekts oder einer Person (Object Detection, ML)
- Erkennen einer Kollision (Collision Avoidance, ML)
- Autonomes Abfahren von mehreren zusammenhängenden Räumen (algorithmische Pfadfindung)

Durch das Kombinieren dieser drei Subkomponenten kann der Roboter als eine Art «Rescue-Search-Roboter» eingesetzt werden. Das Projekt kann in einer Testumgebung für Präsentationen und Infoveranstaltungen genutzt werden. Die Object Detection basiert auf dem SSD MobileNetV2 Modell und die Collision Avoidance auf dem AlexNet.

Durch die stark eingeschränkten Hardware-Eigenschaften, wie zum Beispiel die 4 GB Arbeitsspeicher, musste stark darauf geachtet werden, wann welches Modell verwendet wird und welche Probleme algorithmisch oder mittels Machine Learning gelöst werden.

Die entwickelte Web-App kann zudem verwendet werden, um Pfadfindungsprobleme in der Theorie zu simulieren, um Detailinformationen zu einzelnen realen Roboter-Ausführen zu erhalten oder um die Theorie genauer zu erläutern.

2. Management Summary

2.1 Ausgangslage

Die hier vorliegende Arbeit basiert auf der Machbarkeitsstudie «JetBot: Autonomes und fortlaufendes maschinelles Lernen».

In der Machbarkeitsstudie wird beschrieben, wie man einen Roboter dazu einsetzen kann, um informatikinteressierten Personen einen Einblick in die Informatik, mit Fokus auf die künstliche Intelligenz, zu ermöglichen. Die hier vorliegende Arbeit fokussiert sich auf die Umsetzung dessen.

Die Arbeit umfasst den Aufbau des Roboters sowie der Testumgebung, das Entwickeln zweier Beispielapplikationen, welche mit der Hilfe von künstlicher Intelligenz ein Problem lösen und die Erweiterung der Web-App aus der Machbarkeitsstudie.

2.2 Vorgehen, Technologien

In einem ersten Schritt wurde ein 3D Modell für das Fahrgestell optimiert und gedruckt, der Roboter zusammengebaut und für die Applikationsentwicklung bereit gemacht. Im Anschluss wurden die Applikationen entwickelt sowie die Web-App erweitert.

Der Code ist grössten Teils in Python geschrieben, lediglich für die Ansteuerung der Ultraschallsensoren wird C++ verwendet. Ausgeführt werden die Applikationen über JupyterLab, welches auf dem JetBot läuft.
Für die Objekterkennung wird das Modell SSD MobileNetV2 verwendet, welches über die NVIDIA TensorRT SDK implementiert wird. Die Collision Avoidance basiert auf dem AlexNet welches mittels PyTorch implementiert wird.

Die Web-App basiert auf Node.js, Express, Handlebars und einer lowdb Datenbank Instanz.

2.3 Ergebnisse

Mit den Applikationen ist der Roboter befähigt, sich in Räumen algorithmisch zu bewegen sowie Objekte und Kollisionen mit Hilfe von künstlicher Intelligenz zu erkennen. Ein Modell zur Kollisionserkennung wurde zudem trainiert, Tischkannten zu erkennen. Im Projektvorstellungsvideo werden diese Abläufe aufgezeigt und erklärt.

Über die Web-App kann die Suchapplikation mit der algorithmischen Pfadfindung visualisiert werden und zudem können alte Logdaten der Programmdurchläufe eingesehen werden. Sie soll ferner zur Erläuterung der Theorie dienen.

Mit Hilfe von Performancetests wurde zusätzlich bewiesen, dass der JetBot mit zwei parallellaufenden Machine Learning Modellen an seine Auslastungsgrenzen stösst.

2.4 Ausblick

Es gibt viele weitere Szenarien, in denen der JetBot eingesetzt werden kann und daher auch viele Weiterentwicklungsmöglichkeiten. Das Nvidia Board ist sehr flexibel erweiterbar und kann mit weiteren Komponenten ausgestattet werden. Denkbare Erweiterungen könnten in den folgenden Themengebieten realisiert werden:

Der Roboter könnte, während dem Lösen von Aufgaben, seine Modelle erweitern.

Eine Implementation einer Sprachsteuerung könnte die Interaktion mit dem Roboter stark vereinfachen, die Hardwareeinschränkungen müssten aber stark in die Planung miteinbezogen werden.

Durch einen zweiten Roboter könnte eine Art Kommunikationskanal zwischen den zwei JetBots aufgebaut werden und so ganz neue Aufgaben gelöst werden. Des Weiteren könnten Trainingsdaten oder ganze Modelle zwischen den Robotern ausgetauscht werden.

Da der Roboter das Abfahren von Räumen bereits beherrscht, wäre eine Raum-Mapping Funktion sehr sinnvoll. Hierbei müsste zuerst evaluiert werden, welche weiteren Sensoren gebraucht werden könnten.

Das bereits bestehende Web-App könnte mit mehr Funktionalität ausgestattet werden.

Schlussendlich wären Performance Optimierungen und die Option für mehr Rechenleistung für fast alle Projekte sinnvoll. Hierbei gäbe es Optionen, wie das Hinzufügen eines zweiten Entwicklerboards oder das Anbinden an Cloud Computing Anbietern. Welche dieser Optionen besser ist, müsste in einer weiteren Evaluation begutachtet werden.

3. Einleitung

Die hier vorliegende Arbeit basiert auf der Studienarbeit («JetBot Autonomes und fortlaufendes maschinelles Lernen») [1], die im Herbstsemester 2020 durchgeführt wurde. Nach jener theoretischen Machbarkeitsstudie werden in dieser praxisorientierten Bachelorarbeit JetBot-Applikationen implementiert, welche Einsatzfelder der Informatik mit Fokus auf die Künstliche Intelligenz aufzeigen. Durch die Arbeit wird es möglich sein, neuen Studierenden, die noch am Beginn ihrer Ausbildung stehen, Einsatzfelder in diesem Bereich interaktiv und ansprechend näher zu bringen.

Die folgenden Probleme können von dem JetBot autonom gelöst werden:

- Erkennen eines Objekts oder einer Person
  - Object Detection (ML-Approach)
- Erkennen einer Kollision
  - Collision Avoidance (ML-Approach)
- Autonomes Abfahren von mehreren zusammenhängenden Räumen
  - Algorithmische Pfadfindung
- **Mit der Kombination dieser drei Subkomponenten kann der Roboter für Demonstrationen als eine Art «Rescue-Search-Roboter» eingesetzt werden. Durch das Platzieren des JetBots im Modell kann der Roboter nach einem zuvor definierten Objekt oder einer Person suchen und diese(s) anschliessend, wenn erkannt, anfahren**

Live-Demonstrationen, wie der Roboter diese Probleme löst, können in einer physischen Testumgebung durchgeführt werden. Diese Testumgebung kann für Infoveranstaltungen vor Ort aufgebaut werden.

Des Weiteren können in der eigens entwickelten Web-App Subprobleme wie die Navigation durch Räume visualisiert werden, so dass kein physischer Roboter gebraucht wird. Die Idee hinter dieser Web-App ist es, die theoretischen Grundlagen des Roboters genauer zu erläutern und Detail-Informationen zu physischen JetBot-Ausführungen zur Verfügung zu stellen.

Wie das Web-App funktioniert und wie der Roboter die Probleme löst, kann im Projektvorstellungsvideo nachgeschaut werden.

4. Architektur Übersicht

Die Architektur Übersicht ist in zwei Teile unterteilt. Zuerst wird die Hardwarekonfiguration mit den dazugehörigen Modulen aufgezeigt, anschliessend der Zusammenhang zwischen Hardware und Software genauer erläutert.

4.1 Hardwarekonfiguration

Die Komponenten wurden gemäss der Hardware-Evaluation in der Studienarbeit im Herbstsemester 2020 («JetBot Autonomes und fortlaufendes maschinelles Lernen») [1] bestellt und verbaut. Die folgende Abbildung stellt die Hardware-Architektur logisch, ohne elektrotechnische Details wie Widerstände und Pinouts, dar.

Die grünen Linien sind Verbindungen zu Sub-Komponenten, welche zur Datenübertragung sowie Stromversorgung genutzt werden. Die roten Linien stellen Verbindungen zur reinen Stromversorgung dar.

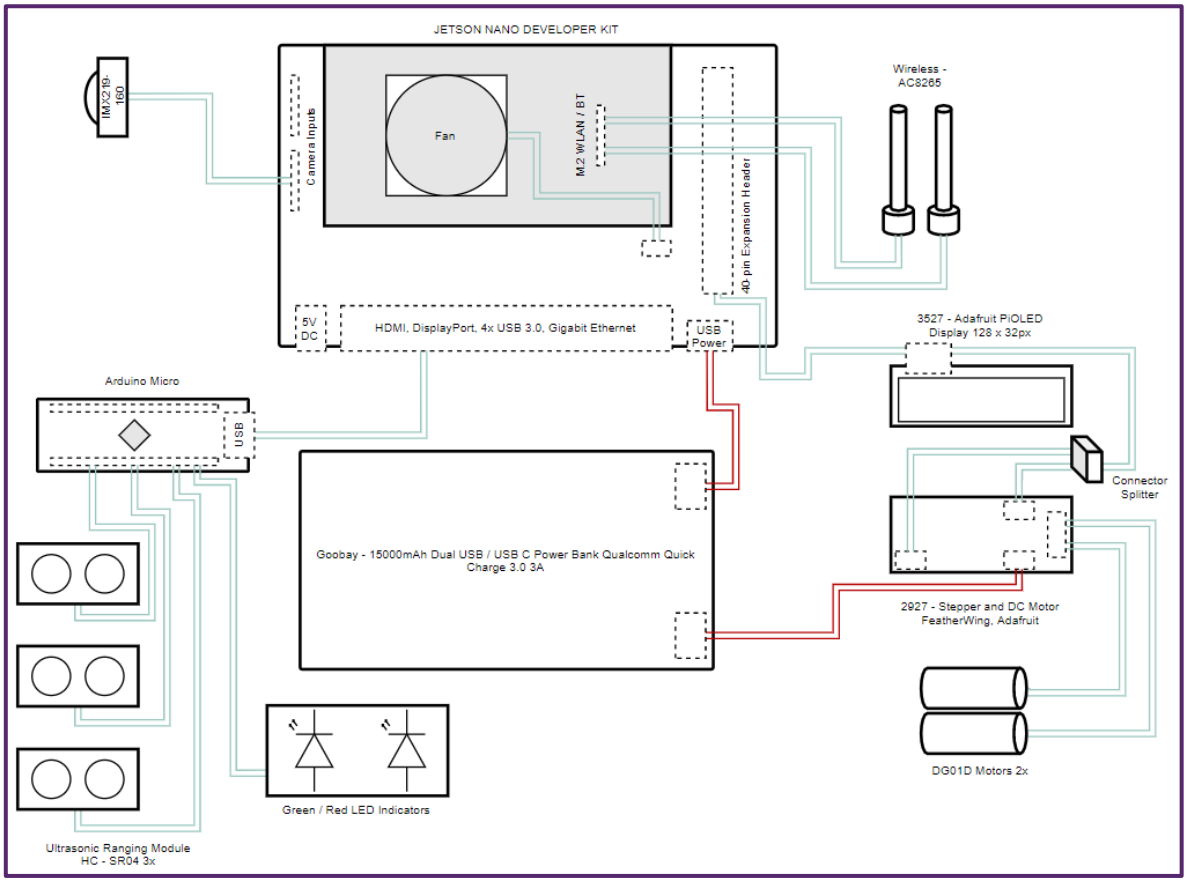


Abbildung 1: Architektur: Hardwarekonfiguration mit dazugehörigen Modulen

An das Nvidia Jetson Nano Developer Kit sind diverse Module angeschlossen. Die Ansteuerung dieser Sub-Komponente erfolgt über diverse Schnittstellen des Developer Kits. Die folgende Tabelle erläutert die jeweiligen Schnittstellen, welche für die Kommunikation genutzt werden.

Tabelle 1: Architektur: Sub-Komponente

Modul:	**Verwendete Schnittstelle(n):**
IMX219-160 Kamera	MIPI-CSI Camera Connector
AC8265 Wireless Modul	M.2 Key E
Arduino Micro Mikrokontroller	USB 2.0, Type A to Micro
HC-SR04 Ultraschall Sensor	2x Digital I/O Pins 1x 5V 1x GND
HC-SR04 Ultraschall Sensor	2x Digital I/O Pins 1x 5V 1x GND
HC-SR04 Ultraschall Sensor	2x Digital I/O Pins 1x 5V 1x GND
Green LED Signallampe	1x Digital I/O Pin 1x GND
Red LED Signallampe	1x Digital I/O Pin 1x GND
3527-Adafruit PiOLED Display	GPIOs
2927- Adafruit Stepper and DC Motor Feather-wing Motor Controller	GPIOs
DG01D Motor	2-pin Connection
DG01D Motor	2-pin Connection
Goobay - 15000mAh Powerbank	2x USB 2, Type A to Micro

Die Standardkomponenten wie der Motor Controller, die zwei Motoren, die WiFi Antennen, das WiFi Modul, das Jetson Nano Developer Kit, das OLED Display, die Kamera und die Powerbank wurden gemäss der offiziellen Nvidia Guidelines verkabelt und montiert [2].

4.1.1 Custom-Components

Da für die Aufgabenstellung weitere Custom-Components wie der Arduino Micro, die drei Ultraschallsensoren und die zwei LEDs notwendig sind, mussten eigene Leiterplatten und Schaltungen entwickelt werden. Das Schema dieser Komponenten sieht wie folgt aus:

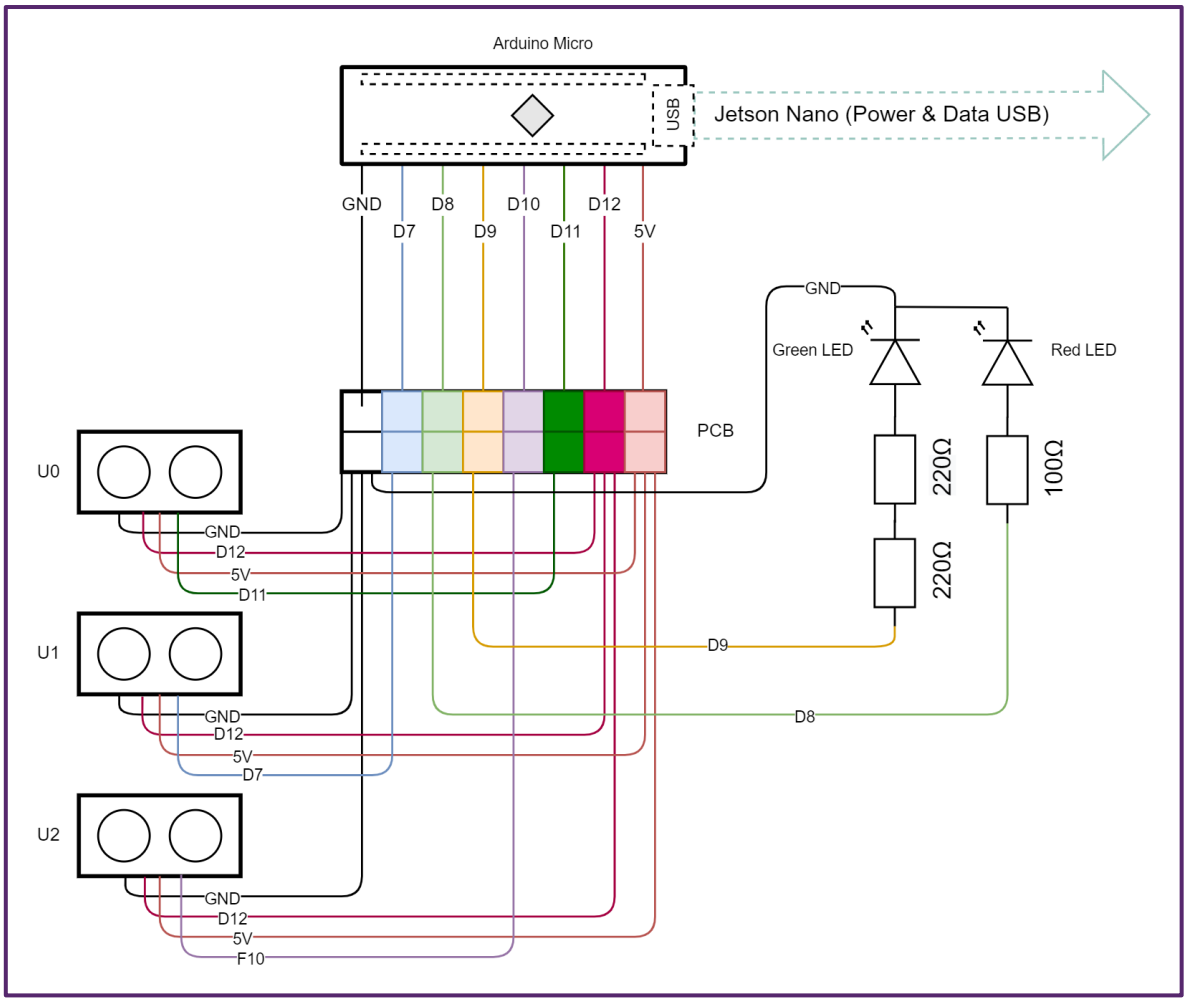


Abbildung 2: Architektur: Schema Custom-Components

Tabelle 2: Architektur: Legende Abbildung 2

U1, U2, U3	Ultraschallsensoren
Arduino Micro	Microcontroller
D7, …, D11	Digitale Pins des Microcontrollers

D12	Gemeinsamer Trigger Pin für alle Ultraschallsensoren (Start-Signal für jeweilige Messung)
PCB	Steckplatine
5V	Stromversorgung
GND	Ground

4.1.2 3D Druck Gehäuse

Um den Roboter für die zu lösenden Probleme optimal aufbauen zu können, wurde auf ein vorgefertigtes Fahrgestell verzichtet. Durch den 3D-Druck eines PLA-Chassis konnten Anpassungen und Optimierungen vorgenommen werden. Das Grundgerüst wurde gemäss der Hardware-Evaluation aus dem Herbstsemester 2020 [1] selektiert, dann aber mittels Design-Anpassungen für die hier bearbeitete Problemstellung optimiert. PLA als Druckmaterial eignet sich optimal für dieses Projekt, da es unkompliziert zu verarbeiten ist und keine grosse Nachbearbeitung braucht.

Die nachfolgenden Abbildungen zeigen diese Anpassungen und ihre jeweiligen Verbesserungen auf:

Tabelle 3: Architektur: 3D Ansicht Fahrgestell

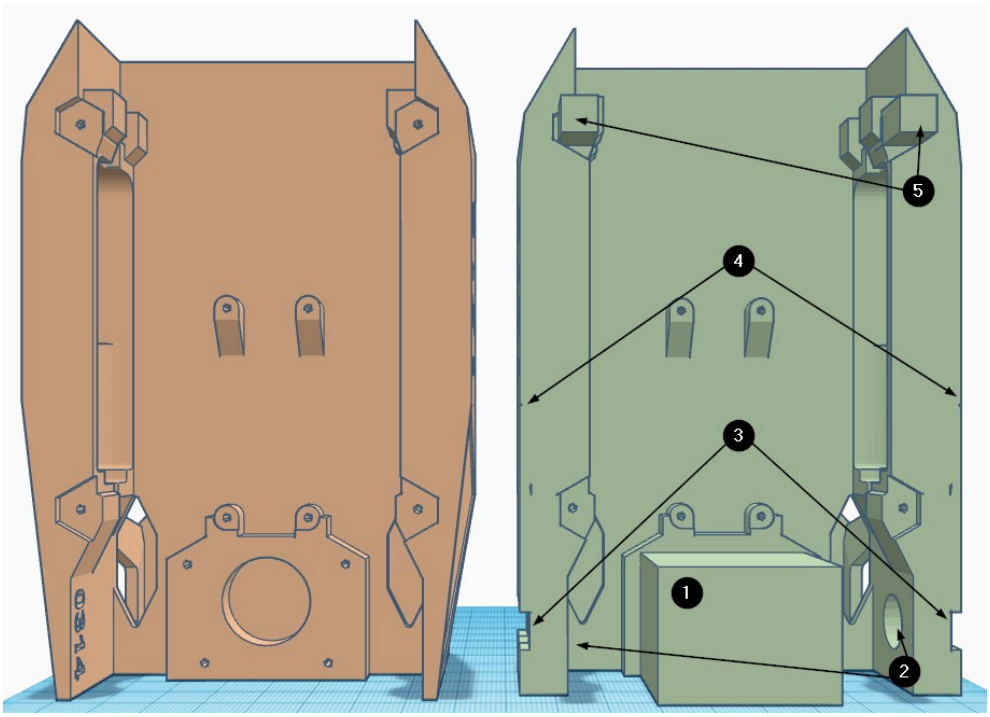
Fahrgestell Ansicht Unten

1) Halterung / Plattform für kugelgelagertes Hinterrad

Abbildung 3: Architektur: Fahrgestell Ansicht Unten

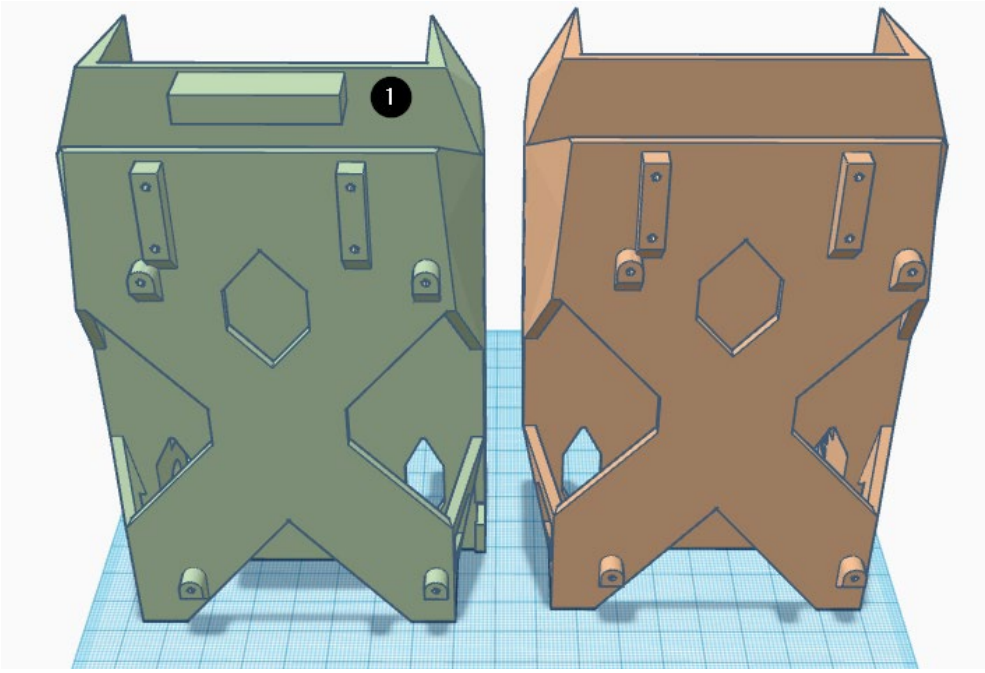
2) Kabelführung links & rechts 3) Vertiefung für Kabelbinder-Platzierung links & rechts 4) Abgeflachte und vergrösserte Aussenkanten links & rechts für Ultraschallsensor, Mikrokontroller und Platinen 5) Montagehilfen für WLAN-Antennen links & rechts
Fahrgestell Ansicht Oben

1) Blockierendes Element, um Stromversorgung fixieren zu können

Abbildung 4: Architektur: Fahrgestell Ansicht Oben

Tabelle 4: Architektur: Legende Tabelle 2

	Original 3D Design
	Optimiertes 3D Design

4.2 Zusammenhang Hardware und Software

Die Interaktion mit dem Jetson Nano Developer Kit sowie Entwicklung erfolgt über Jupyter Notebook und der Programmiersprache Python 3 [1].

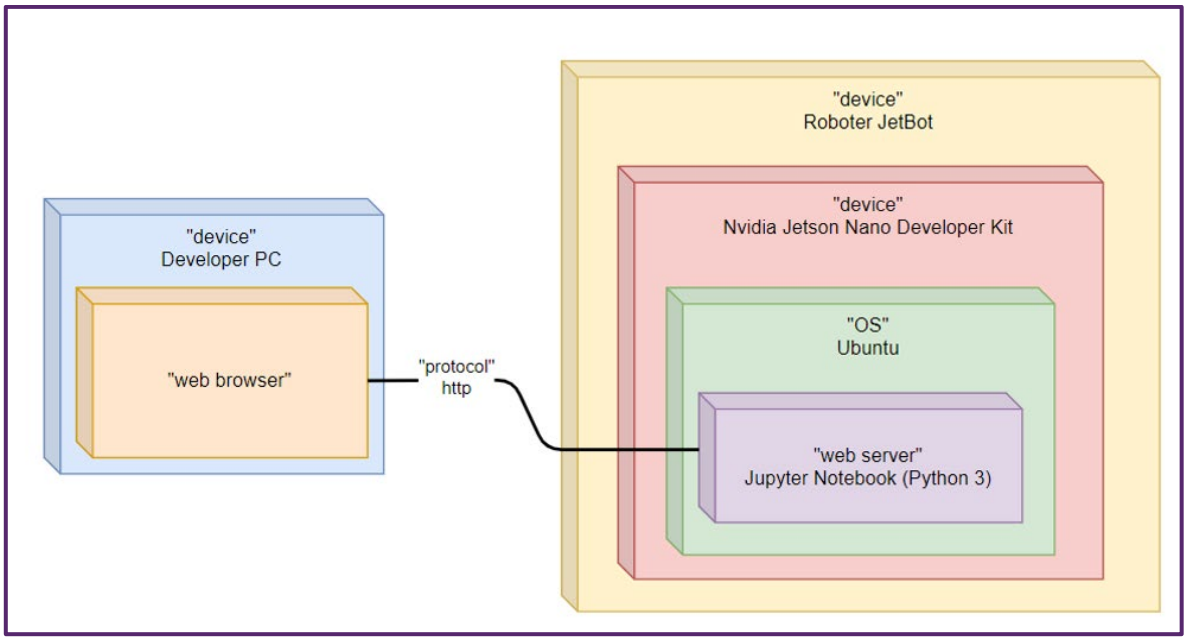


Abbildung 5: Architektur: Zusammenspiel Hardware und Software [1]

4.2.1 Zusammenhang Arduino Micro und Nvidia Jetson Nano

Der Mikrokontroller Arduino Micro und die Ansteuerung der zwei Ultraschallsensoren erfolgt über die Programmiersprache C++. Der Mikrokontroller wird verwendet, um die Sensoren in Echtzeit ansteuern zu können und um Timing- beziehungsweise Scheduling-Issues, welche beim direkten Anschluss an den Jetson Nano auftreten könnten, zu vermeiden [3]. Die Datenübertragung zwischen dem Mikrokontroller und dem Jetson Board erfolgt seriell über USB 2. Die weitere Datenverarbeitung erfolgt auf dem Nvidia Jetson Nano mittels Python 3.

Abbildung 6: Architektur: Jetson Nano - Arduino Micro

4.2.2 Ansteuerung Sensoren und Datenübertragung Arduino Micro

Das folgende Flowchart zeigt die Umsetzung des Programmcodes auf dem Arduino Micro auf. Das Programm startet, sobald der Mikrokontroller an die Stromversorgung angeschlossen wird. Wichtig zu beachten ist, dass der Code, welcher für die Ansteuerung und Übertragung der Daten verantwortlich ist, in einer Schleife ausgeführt wird. Diese Schleife wird nur durch einen Stromunterbruch oder einen User Interrupt auf dem Mikrokontroller selbst beendet.

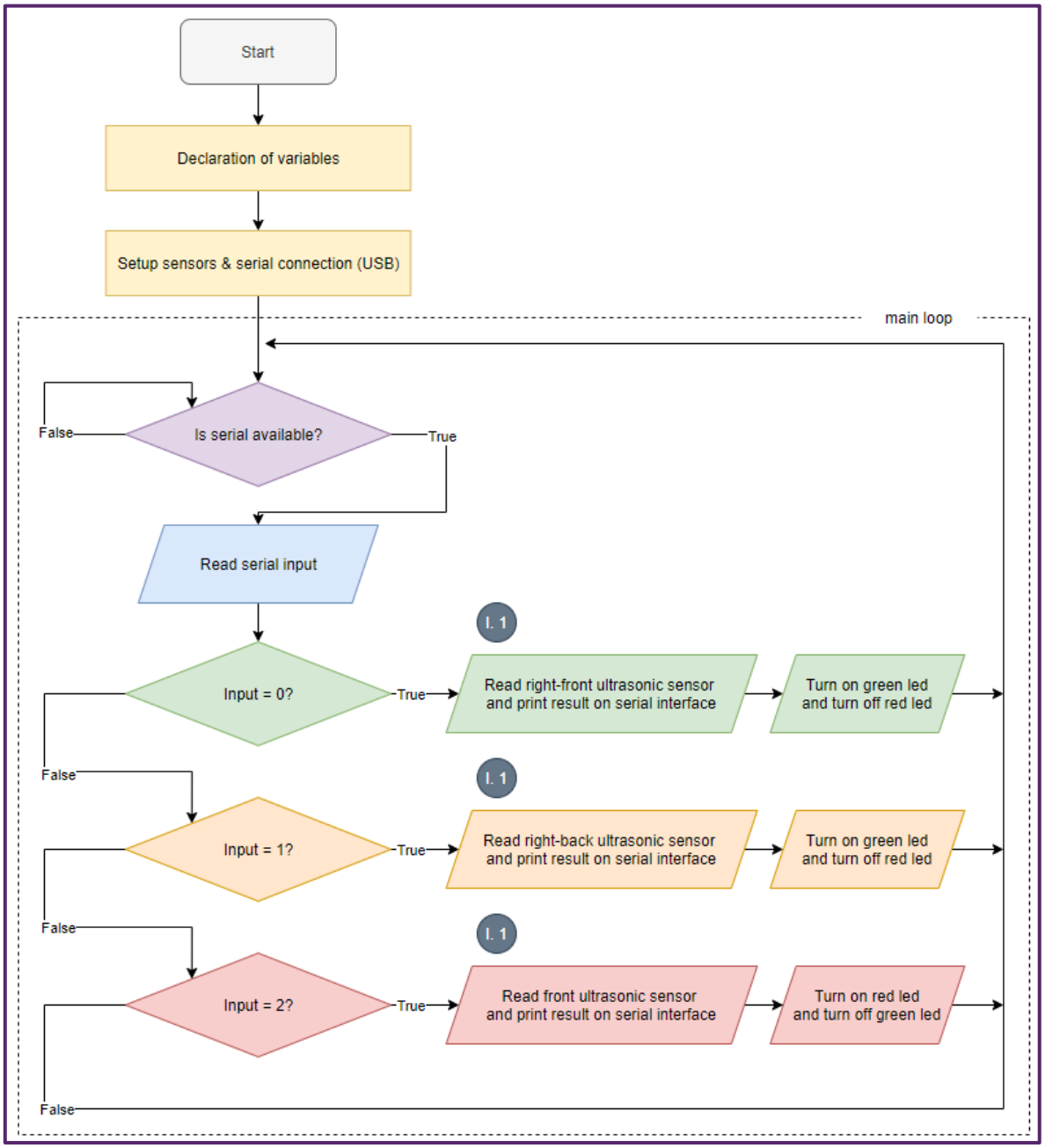


Abbildung 7: Architektur: Flowchart Arduino Micro

Tabelle 5: Architektur: Legende Abbildung 7

I. 1	Die Abfrage der Ultraschallsensoren gibt die Echo Time (in µs) zurück, die dem Median von 10 Signalen entspricht. Somit wird die Messfehlerwahrscheinlichkeit reduziert. $$\boldsymbol{ReturnValue(EchoTime)} = \frac{\boldsymbol{EchoTime}_{\frac{10}{2}} + \boldsymbol{EchoTime}_{\frac{10}{2}+1}}{\mathbf{2}}$$ **Messgenauigkeit**: ***«Die systembedingte Messgenauigkeit beträgt ca. 3mm und hängt mit der internen Abtastrate des Moduls zusammen. Ein weiterer Faktor ist die Temperaturabhängigkeit der Schallgeschwindigkeit in [der] Luft.» [4].***

Sampling-Rate Arduino Main-Loop

Um zu erfahren, wie viele Durchgänge des Main-Loops pro Sekunde durchlaufen werden, ist ein Testprogramm[1] mit einem Counter implementiert worden. Das Programm misst jeweils für eine Sekunde lang die Durchlaufanzahl und wird zehn Mal ausgeführt. Die folgende Tabelle beinhaltet die Resultate der Messung:

Tabelle 6: Architektur: Sampling-Rate Arduino

Messungen	**M1**	**M 2**	**M 3**	**M 4**	**M 5**	**M 6**	**M 7**	**M 8**	**M 9**	**M 10**
Anzahl Durchläufe	19509	19502	19505	19508	19499	19509	19502	19512	19510	19497
Mittelwert	$\bar{x} = \frac{1}{n}\sum_{k=1}^{n} M_k \approx 19505$									

Der Main-Loop des Mikrokontrollers wird etwa 19'505 / Sekunde durchlaufen.

[1] https://github.com/yV11/ba-fs2021/blob/main/ArduinoLibrary/main-loop-sampling-rate/main-loop-sampling-rate.ino

5. Performance Evaluation

Durch die limitierten Hardware-Ressourcen des Jetson Nano ist es wichtig, Machine Learning Modelle zu optimieren und nur dann zu verwenden, wenn sie für die Problemlösung gebraucht werden. Das Nvidia Jetson Nano Developer Kit hat folgende Hardwareeigenschaften:

Tabelle 7: Performance: Nvidia Jetson Nano, technische Spezifikationen [5]

GPU:	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores
CPU:	Quad-core ARM Cortex-A57 MPCore processor
RAM:	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Disk:	SD Card, 64GB, Class 10
Swap:	2 GB
Bemerkung:	Um zusätzliche Ressourcen freizumachen, wurde das Ubuntu-GUI deaktiviert. Dieses bleibt auch nach der Performance Evaluation deaktiviert.

5.1 Tests und Benchmarks

Die folgenden Experimente wurden durchgeführt, um aufzuzeigen, wie sich die Performance im Verhältnis zu aktiven ML-Modellen verändert.

5.1.1 Testumgebung

- Um die Experimente überwachen zu können, wird das «jetson_stats» Python 3 Modul verwendet [6]
  - Während den Experimenten werden die Werte in ein CSV-Log geschrieben. Die folgenden Daten wurden als relevant eingestuft, da sie einen Einfluss auf die Performance der ML-Modelle haben:
    - CPU1, CPU2, CPU3, CPU4 (alle Cores)
    - GPU
    - RAM
    - Temp CPU
    - Temp GPU
  - Jedes Experiment wird für 10 Minuten überwacht und ausgeführt

Performance Baseline

Die folgenden Daten dienen als Richtwerte. Sie stellen die Performance des Roboters direkt nach dem Aufstarten ohne weitere Last dar.

Tabelle 8: Performance: Variablen Baseline

Dauer:	10 Minuten (600 Sekunden)
Log Einträge:	1029

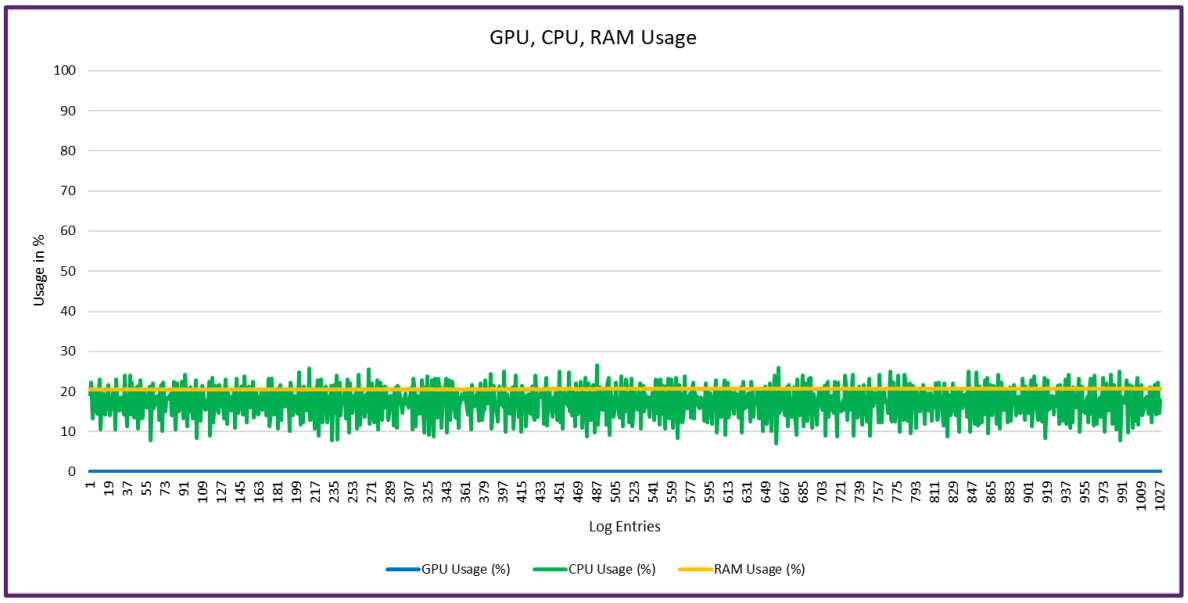


Abbildung 8: Performance: Auslastung in %, Baseline

- Arithmetisches Mittel, GPU-Auslastung: 0.00%
- Arithmetisches Mittel, CPU-Auslastung: 17.51%
- Arithmetisches Mittel, RAM-Auslastung: 20.62%

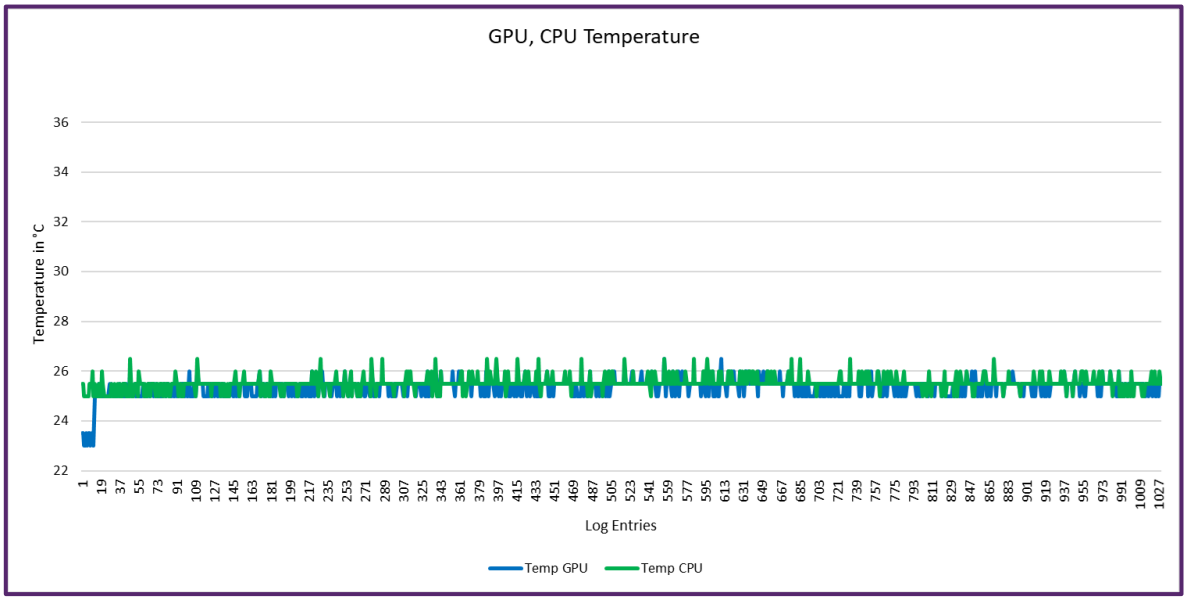


Abbildung 9: Performance: Temperaturen, Baseline

- Arithmetisches Mittel, GPU-Temperatur: 25.35°C

- Arithmetisches Mittel, CPU-Temperatur: 25.52°C

5.1.2 Testdurchläufe und Resultate

Im ersten Experiment wird nur das Object Detection Modell ausgeführt. Im zweiten Experiment wird zusätzlich noch das Collision Avoidance Modell ausgeführt.

Performance Experiment 1

Die folgende Tabelle zeigt die Parameter für das Object Detection Modell auf:

Tabelle 9: Performance: Variablen Experiment 1

Modell:	SSD MobileNetV2
Kamera Auflösung:	1280 x 720px
Dauer:	10 Minuten (600 Sekunden)
Log Einträge:	1029

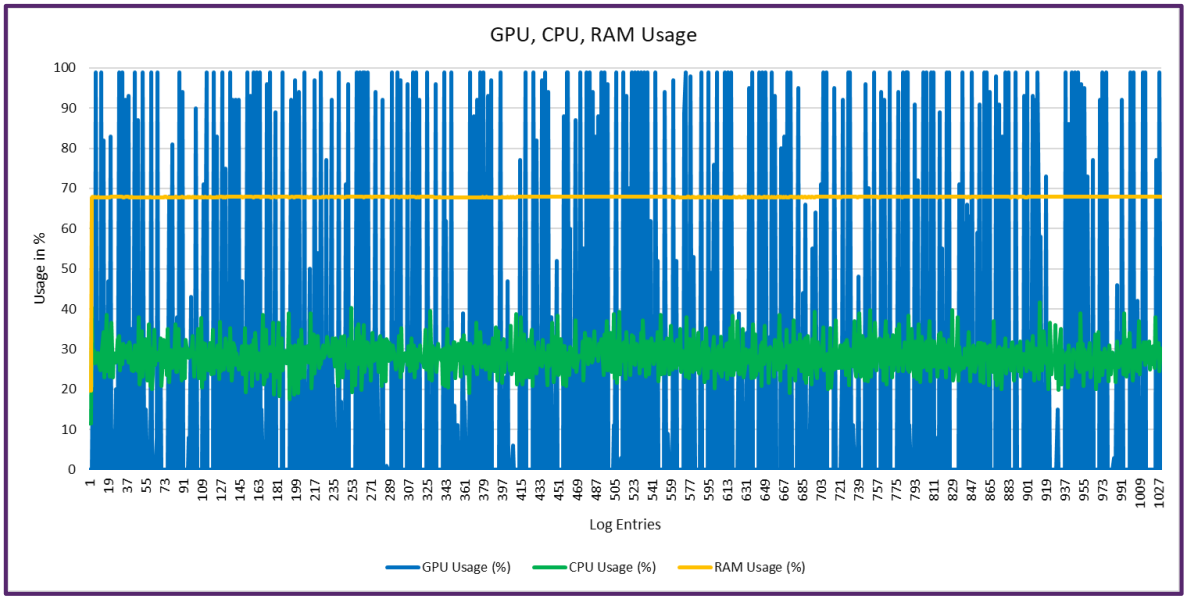


Abbildung 10: Performance: Auslastung in %, Experiment 1

- Arithmetisches Mittel, GPU-Auslastung: 20.17%
  - Delta zu Baseline: $\Delta \approx +20.17\%$
  - Die durchschnittliche GPU-Auslastungszunahme ist immer noch vertretbar**.** Dennoch sind viele Peaks ersichtlich, welche eine Auslastung von bis zu 100% darstellen. Dies könnte zu einem späteren Zeitpunkt zu Problemen führen, wenn weitere ML-Modelle ausgeführt werden
- Arithmetisches Mittel, CPU-Auslastung: 28.30%
  - Delta zu Baseline: $\Delta \approx +10.79\%$

- Die durchschnittliche CPU-Auslastungszunahme ist immer noch tief. Es sind keine Peaks ersichtlich und die Daten weissen eine relativ kleine Spannweite auf

- Arithmetisches Mittel, RAM-Auslastung: 67.76%
  - Delta zu Baseline: $\Delta \approx +47.14\%$
  - Die durchschnittliche RAM-Auslastungszunahme ist bereits hoch. Mit einer konstanten RAM-Auslastung von bereits knapp 70% können bei weiteren ML-Modellen Probleme auftreten

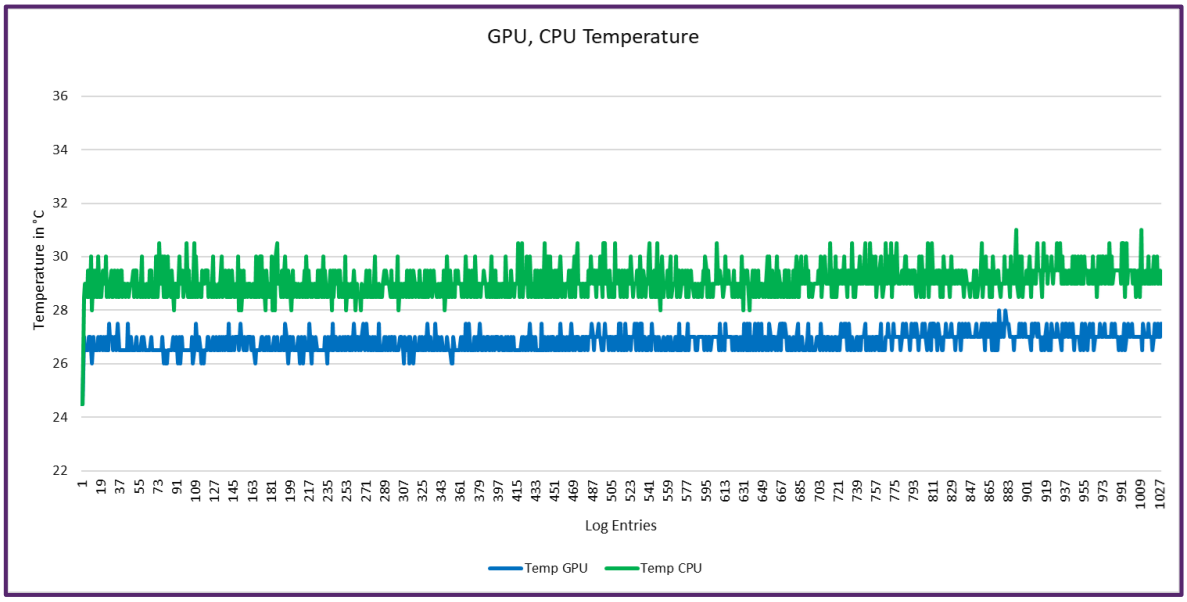


Abbildung 11: Performance: Temperaturen, Experiment 1

- Arithmetisches Mittel, GPU-Temperatur: 26.82°C
  - Delta zu Baseline: $\Delta \approx +1.47℃$
  - Die GPU-Temperaturerhöhung von knapp 1.5°C stellt kein Problem dar
- Arithmetisches Mittel, CPU-Temperatur: 29.08°C
  - Delta zu Baseline: $\Delta \approx +3.56℃$
  - Die CPU-Temperaturerhöhung von knapp 1.5°C stellt kein Problem dar

Performance Experiment 2

Die Parameter für das Object Detection Modell wurden wie beim Experiment 1 gesetzt. Zusätzlich wurden folgende Parameter für das Collision Avoidance Modell definiert:

Tabelle 10: Performance: Variablen Exmperiment 2

Modell:	SSD MobileNetV2 (Object Detection) AlexNet (Collision Avoidance)
Kamera Auflösung:	1280 x 720px

Dauer:	10 Minuten (600 Sekunden)
Log Einträge:	1029

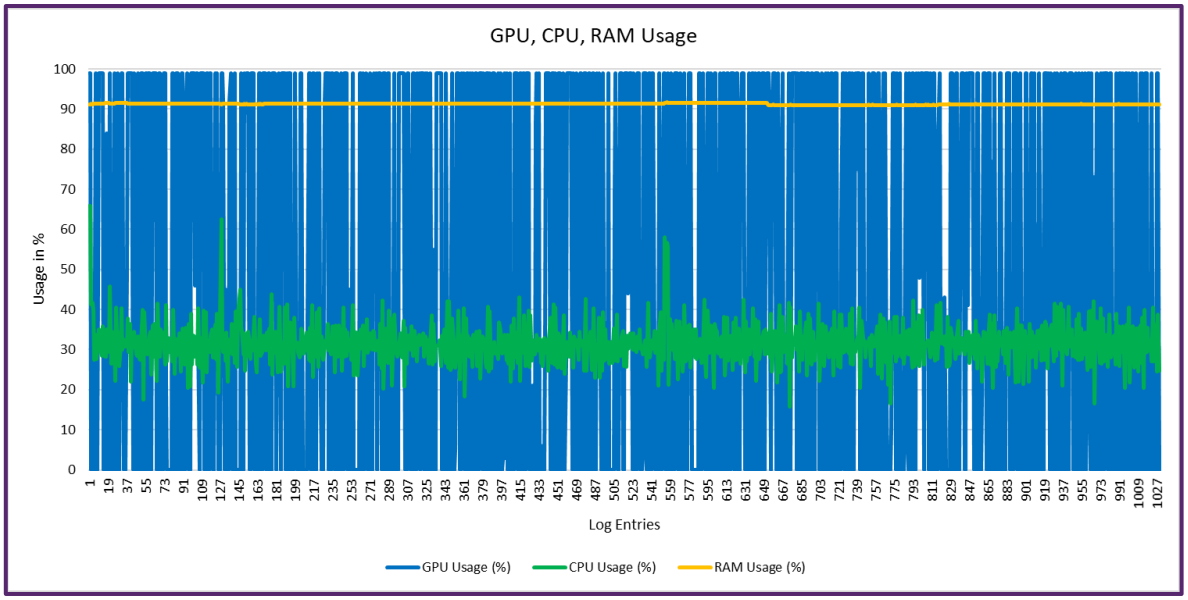


Abbildung 12: Performance: Auslastung in %, Experiment 2

- Arithmetisches Mittel, GPU-Auslastung: 46.68%
  - Delta zu Experiment 1: $\Delta \approx +26.51\%$
  - Die durchschnittliche GPU-Auslastungszunahme ist immer noch vertretbar, aber es sind noch einmal deutlich mehr Peaks im Vergleich zum Experiment 1 vorhanden, was problematisch bei zukünftigen Projekten sein könnte
- Arithmetisches Mittel, CPU-Auslastung: 31.64%
  - Delta zu Experiment 1: $\Delta \approx +3.34\%$
  - Die durchschnittliche CPU-Auslastungszunahme ist immer noch tief und die Auslastung hat keine grosse Spannweite, daher stellen diese Werte kein Problem dar
- Arithmetisches Mittel, RAM-Auslastung: 91.28%
  - Delta zu Experiment 1: $\Delta \approx +23.52\%$
  - Die durchschnittliche RAM-Auslastung ist nun bei über 90%. Ein drittes Modell mit dieser Konfiguration parallel auszuführen, ist nicht mehr möglich

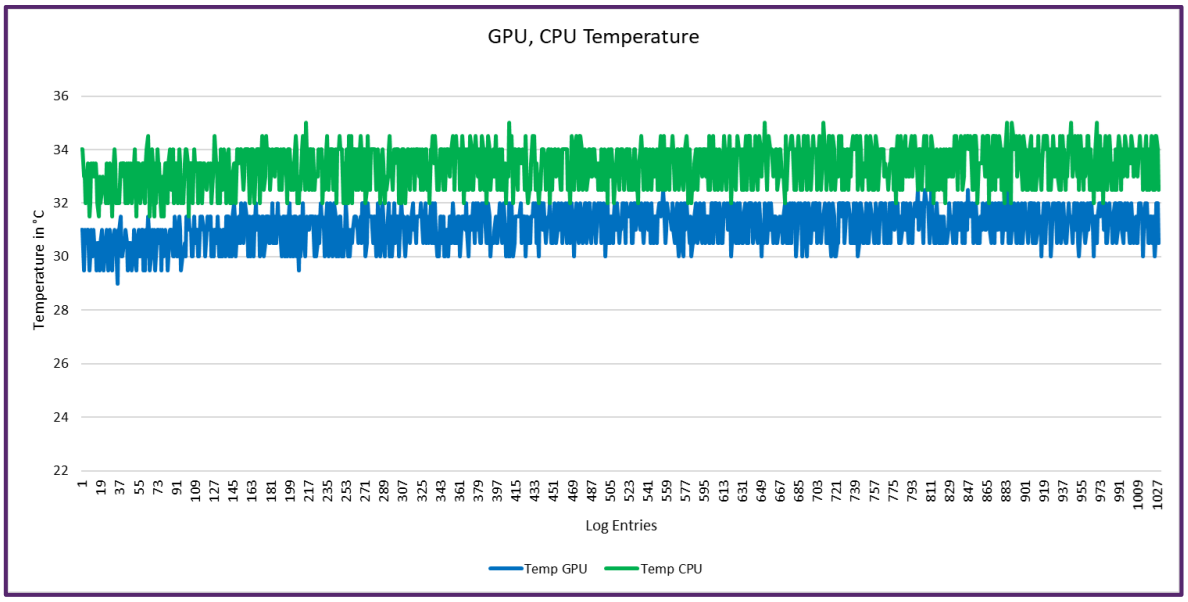


Abbildung 13: Performance: Temperaturen, Experiment 2

- Arithmetisches Mittel, GPU-Temperatur: 31.12°C
  - Delta zu Experiment 1: $\Delta \approx +4.30℃$
  - Auch die GPU-Temperaturerhöhung von 4.30°C stellt kein Problem dar
- Arithmetisches Mittel, CPU-Temperatur: 33.43°C
  - Delta zu Experiment 1: $\Delta \approx +4.38℃$
  - Auch die CPU-Temperaturerhöhung von knapp 4.4°C stellt kein Problem dar

5.2 Fazit

Die folgende Abbildung veranschaulicht die Resultate der Baseline sowie die Durchläufe der Experimente 1 und 2.

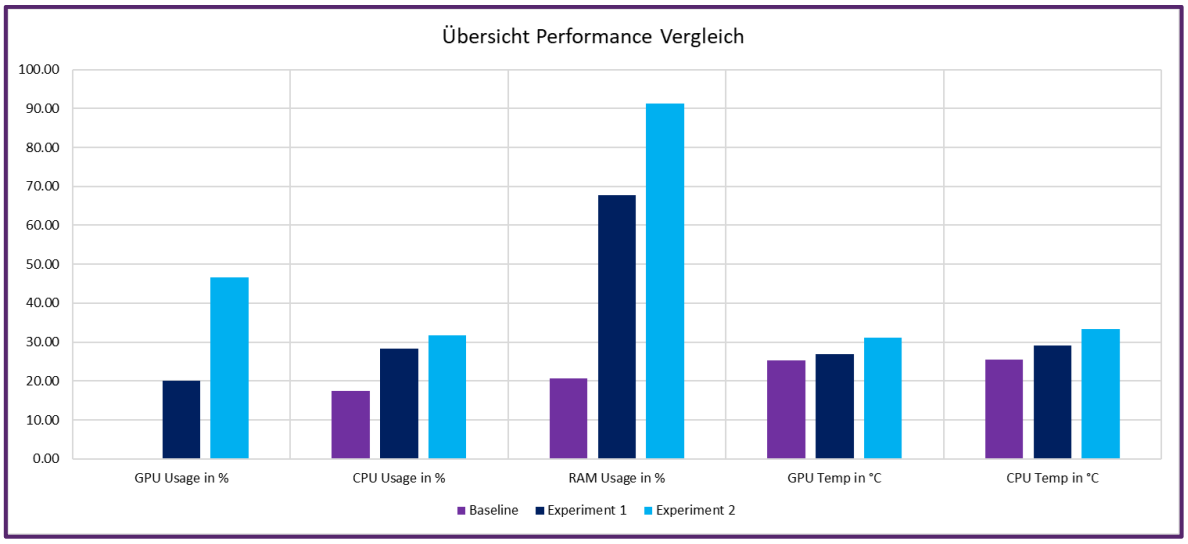


Abbildung 14: Performance: Fazit

5.2.1 GPU-Auslastung

Im Durchschnitt ist die GPU-Auslastung nie über 50% Auslastung gestiegen. Bereits im Experiment 1 sind aber viele Peaks von bis zu 100% Auslastung aufgetreten, im Experiment 2 (mit zwei Modellen) ist die Anzahl der Peaks noch einmal deutlich gestiegen und die Auslastung war nie konstant.

Die Temperaturzunahme der GPU ist klein und hat kein Thermal Throttling ausgelöst.

Ein weiteres (drittes) Modell, welches die GPU beanspruchen würde, wäre bereits problematisch.

5.2.2 CPU-Auslastung

Die CPU-Auslastung hat sich mit der Anzahl der Modelle leicht erhöht, ist aber immer konstant. Mit einer maximalen durchschnittlichen Auslastung von 31.64% ist eine Reserve für weitere Tasks vorhanden.

Die Temperaturzunahme der CPU ist klein und hat kein Thermal Throttling ausgelöst.

5.2.3 RAM-Auslastung

Die Auslastung des Arbeitsspeichers nahm mit der Anzahl parallellaufenden Modellen deutlich zu. Beim letzten Experiment (Experiment 2) war die durchschnittliche Auslastung bei bereits 91.28%, was keinen Spielraum für weitere Modelle zulassen würde.

Die 4GB Arbeitsspeicher sind für das Ausführen von mehreren ML-Modellen sehr limitierend und ein drittes Modell parallel auszuführen ist nicht mehr möglich.

6. Ansteuerung der Motoren und Implementation der algorithmischen Pfadfindung

Der Roboter navigiert sich algorithmisch durch die zusammenhängenden Räume. Dies wird mithilfe von drei Ultraschall-Sensoren ermöglicht, von denen zwei seitlich montiert sind, sowie von einem Sensor, der die Distanz nach vorne misst. Auf einen Machine Learning Ansatz zur Pfadfindung wurde verzichtet, um die Auslastung der bereits limitierten Hardware nicht weiter in die Höhe zu treiben, sodass die Modelle zur Collision Avoidance und Object Detection die maximal mögliche Performance zur Verfügung haben, wie bereits im vorherigen Kapitel beschrieben.

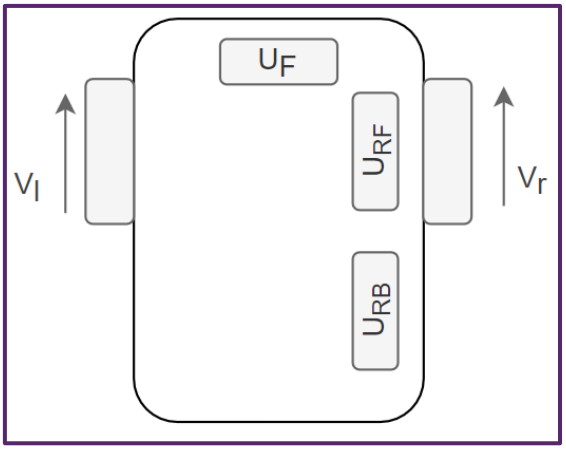


Abbildung 15: Pfadfindung: Navigation Hardware

Der Roboter besitzt zwei Motoren auf der Vorderachse (V_l & V_r) sowie eine kugelgelagerte Kugel, welche auf der Hinterachse zentriert montiert ist. Diese dient nur als Unterstützung und hat keinen aktiven Lenkeinfluss und somit auch keine Ansteuerungsmöglichkeit.

6.1 Algorithmische Pfadfindung

Der Roboter wird in einem zuvor definierten Startraum nahe an einer Wand positioniert, die Wand muss sich zudem auf der rechten Seite des Roboters befinden (Precondition). Durch die zwei rechten Ultraschallsensoren, U_{RB} und U_{RF}, erkennt der Roboter, wie nahe er sich an der Wand befindet.

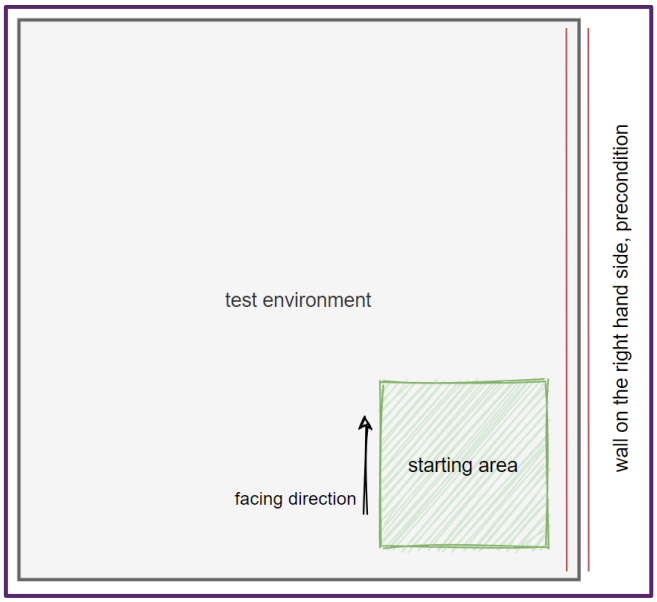


Abbildung 16: Pfadfindung: Platzierung des Roboters im Modell

6.1.1 Follow-Wall

Um die verschiedenen Räume abfahren zu können, wird ein Follow-Wall Algorithmus[2] verwendet. Mit diesem ist es möglich, Wänden parallel zu folgen. So kann sichergestellt werden, dass alle Räume erreicht werden. Das folgende Diagramm stellt den Ablauf des Algorithmus dar:

[2] https://github.com/yV11/ba-fs2021/tree/main/AlgorithmLibrary

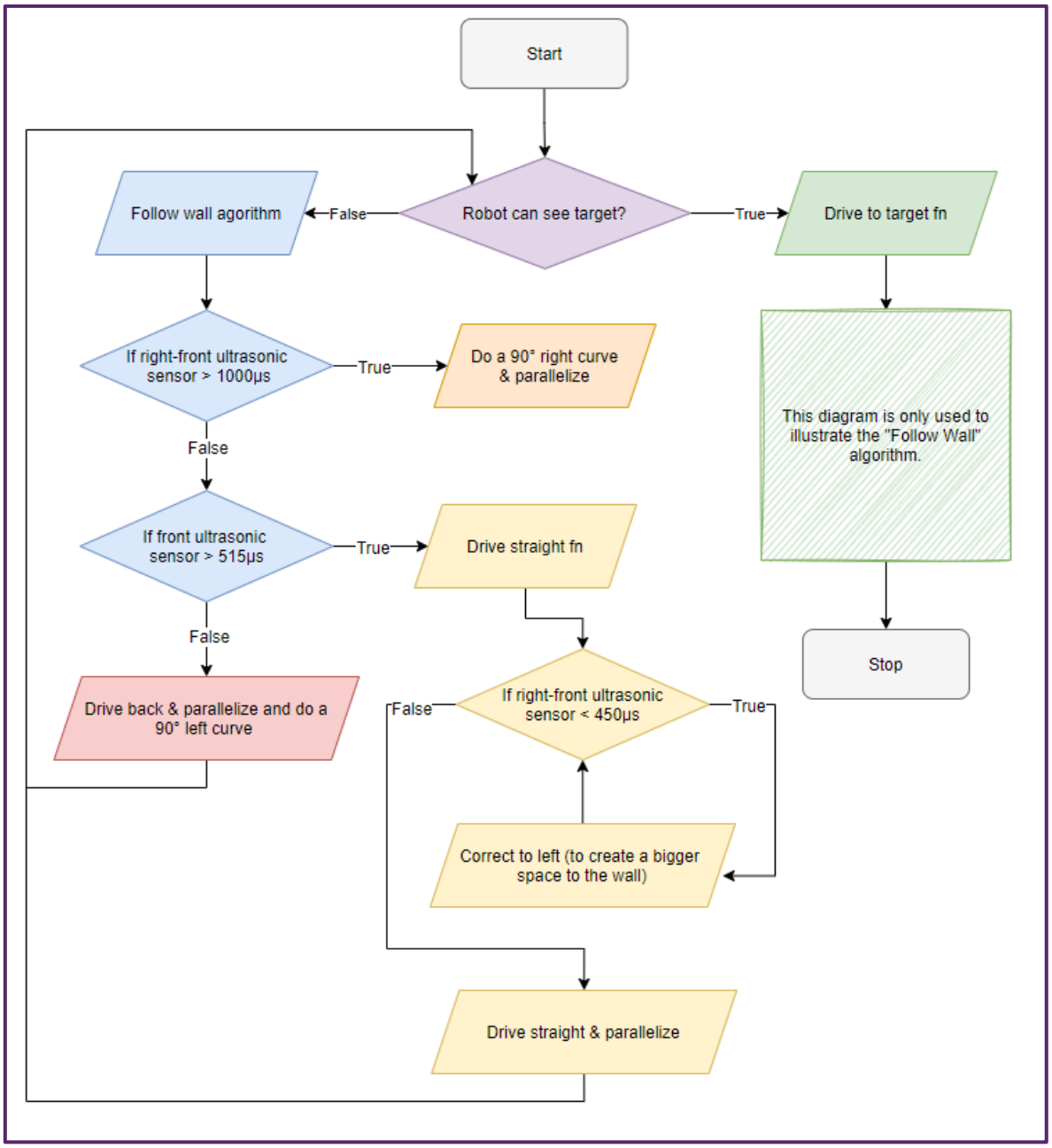


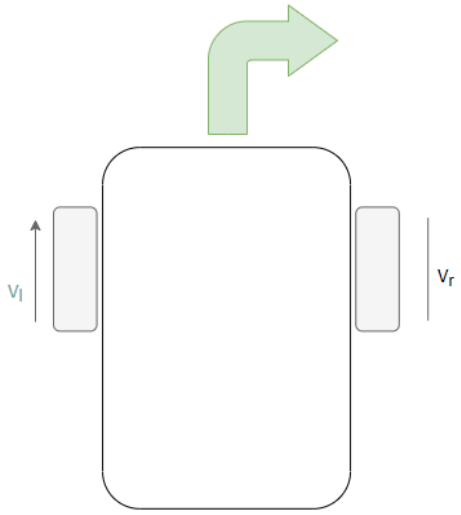
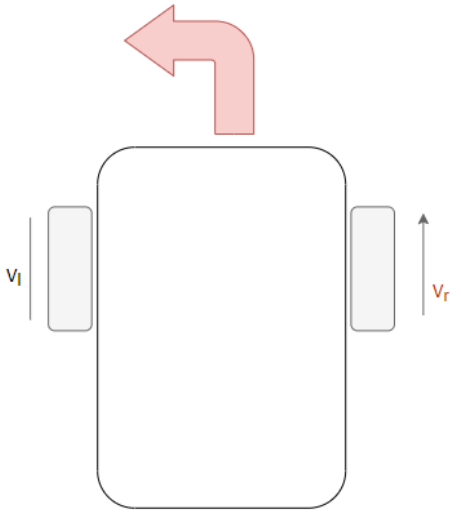
Abbildung 17: Pfadfindung: Follow-Wall

Ein weiterer wichtiger Faktor ist, dass der Roboter tendenziell den linken Motor (V_l) leicht stärker ansteuert ($MotorPowerV_l = MotorPowerV_R + 0.05$). Das hat zur Folge, dass der Roboter sich nie zu weit von der Wand entfernt. Durch diese Eigenschaft fährt der Roboter ein leichtes Zick-Zack Muster.

6.1.2 Kurvenfahrt

Die folgende Tabelle dient als Erläuterung, wie Kurven gefahren werden:

Tabelle 11: Pfadfindung: Kurvenfahrt

 *Abbildung 18: Pfadfindung: Kurvenfahrt, Rechts*	 *Abbildung 19: Pfadfindung: Kurvenfahrt, Links*
Rechtskurve: Der linke Motor (V_l) gibt Schub nach vorne.	**Linkskurve:** Der rechte Motor (V_r) gibt Schub nach vorne.

Optimierungsvorschlag Kurvenfahrt

Die Kurvenfahrt könnte durch einen zusätzlichen Kompass optimiert werden. Dadurch wäre es möglich, 90° Kurven präzise abzufahren, da das Feedback des Sensors als Referenz gebraucht werden könnte. Ein solcher Sensor wäre zum Beispiel das «Triple-axis Accelerometer+Magnetometer (Compass)» Board vom Hersteller Adafruit [7]. Es wurde aber trotzdem darauf verzichtet, da die Lösung auch mit den zu Verfügung gestellten Komponenten erreicht werden konnte und so die Komplexität der Ansteuerung der Sensoren kleiner gehalten werden konnte.

7. Object Detection

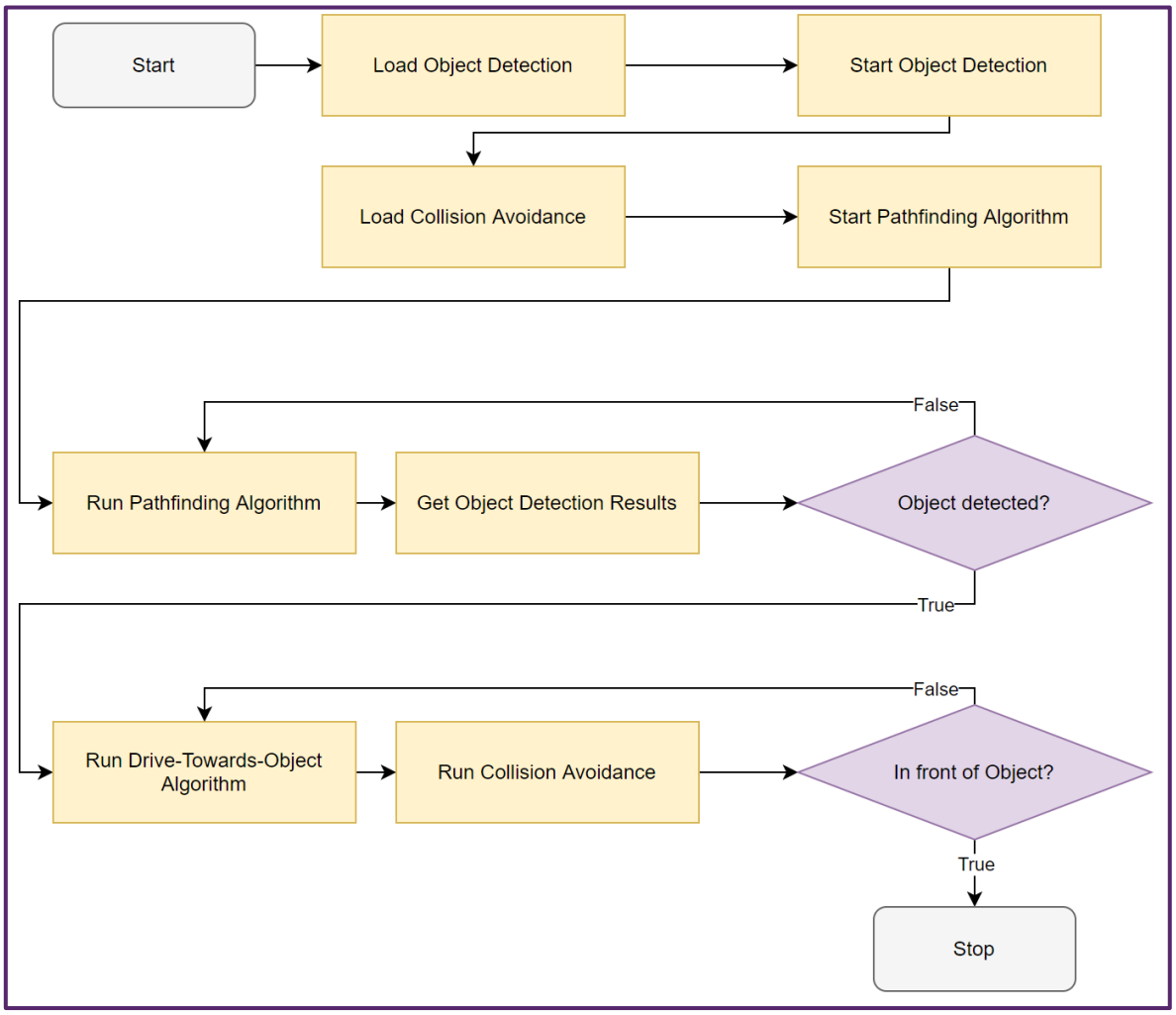


Abbildung 20: Object Detection: Ablaufdiagramm

Für die Object Detection wird basierend auf der Evaluation aus der letzten publizierten Arbeit SSD MobileNetV2 verwendet.

Die Object Detection wird in einem eigenen Thread gestartet. Dieser Thread stellt sicher, dass der Videostream inklusive Object Detection flüssig im JupyterLab dargestellt wird. Die Daten der Object Detection können, über Instanzvariablen, durch den Main-Thread ausgelesen werden. Die Object Detection läuft vom Start bis zum Ende des Programms, wie es im Ablaufdiagramm oben zu sehen ist, sie hat jedoch indirekt unterschiedliche Anwendungsbereiche.

Nach dem Start der Applikation wird ein gewünschtes Objekt aus dem Datenset gewählt und danach wird die Object Detection gestartet und damit das Modell geladen. Zusätzlich wird auch das Modell für die Collision Avoidance geladen, um Verzögerungen während des Programmdurchlaufs zu vermeiden. Die Modelle müssen nur beim Start des Programms einmal geladen werden und können anschliessend mehrmals in den Programmdurchläufen verwendet werden.

Sobald das Modell geladen wurde, startet die Algorithmische Pfadfindung. Die Pfadfindung läuft so lange, bis das gewählte Objekt erkannt wird.

Anschliessend wird die Collision Avoidance und der «Drive-Towards-Object» Algorithmus gestartet. Hier fungieren die Ergebnisse der Object Detection nicht nur als Erkennungsmittel, sondern auch für die Berechnung der Ausrichtung des Roboters.

Sobald die Collision Avoidance anschlägt, wird die Object Detection und das komplette Programm beendet. In diesem Fall steht der Roboter vor seinem Zielobjekt.

In den folgenden Unterkapiteln wird das Modell und das neuronale Netz genauer erklärt, welche zusammen SSD MobileNetV2 bilden. Zusätzlich werden auch die Daten, mit welchen es trainiert wurde, und der Anfahralgorithmus erläutert.

7.1 Modell

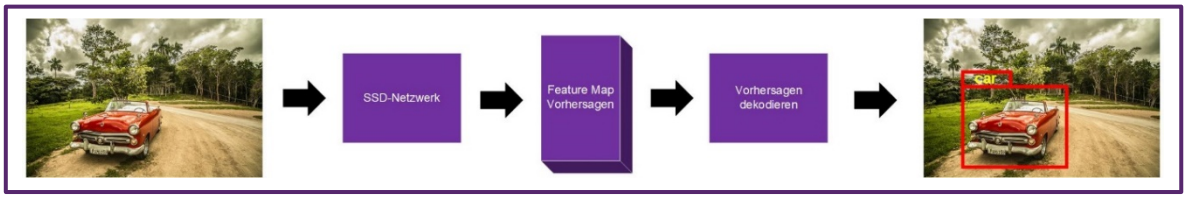


Abbildung 21: Object Detection: SSD (Bild Oldtimer: [8])

Als Modell für die Object Detection wurde SSD (Single Shot MultiBox Detector) in der letzten publizierten Arbeit evaluiert. Es wird mittels der NVIDIA TensorRT SDK implementiert, diese ist eine Hochleistungsfähige Deep-Learning-Inferenz aufgebaut auf CUDA. CUDA ist eine Technologie und Programmierschnittstelle, welche von NVIDIA entwickelt wurde, um die Rechenleistung von GPUs für Applikationen nutzbar zu machen. Somit wurde das Modell auch für den JetBot optimiert, welcher den grössten Teil seiner Leistung von seiner GPU bezieht. [9]

SSD ist ein auf Regression (Single Shot) basierendes Modell. Für die Regression der Begrenzungsrahmen (Bounding Boxes) wird eine MultiBox Technik verwendet. Das eigentliche Klassifizieren der Objekte geschieht mit Hilfe eines Detectors. Die Vorhersagen werden anhand von Multiskalen-Features zusammengeführt. Dies geschieht durch ein «single stage object detection network». Um Vorhersagen aus den Feature-Maps zu erstellen, wird auf dem gegebenen Bild ein Deep-Learning-CNN ausgeführt. Die Vorhersagen werden dann vom Detector gesammelt und decodiert, um dann daraus Begrenzungsrahmen zu generieren. [1] [10]

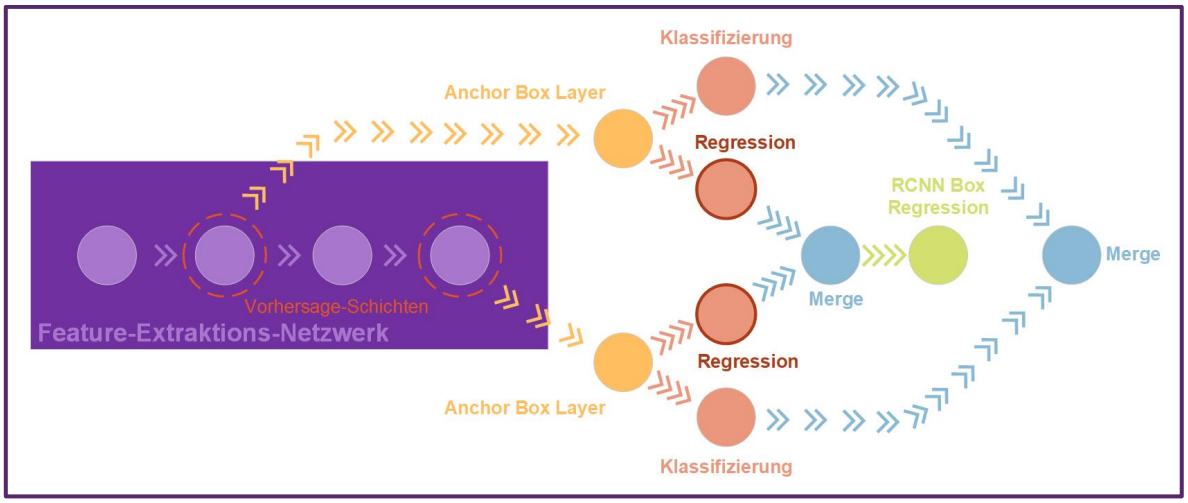


Abbildung 22: Object Detection: SSD Netzwerk

7.1.1 Feature-Extraktions-Netzwerk

In diesem Netzwerk geschieht ein Prozess der Dimensionalität-Reduktion, in welchem ein anfänglicher Satz von Bilddaten auf handlichere Gruppen für die Verarbeitung reduziert wird. Die grossen Datensätze definieren sich oft mit ihrer grossen Anzahl von Variablen, deren Verarbeitung viele Rechenressourcen erfordern. Um die verarbeitende Datenmenge effektiv zu reduzieren und den ursprünglichen Datensatz trotzdem weiterhin exakt und vollständig zu beschreiben, werden Feature-Extraktionsmethoden verwendet, welche Variablen zu Merkmalen auswählen und kombinieren.

7.1.2 Vorhersage-Schichten

Die Vorhersage-Schichten werden aus dem Feature-Extraktions-Netzwerk gewählt. Es kann jede Ebene aus dem Netzwerk als Vorhersageebene gewählt werden. Um hier jedoch die Vorteile der Multiskalen-Features zu nutzen, welche bei der Objekterkennung Anwendung finden, werden hier Feature-Maps unterschiedlicher Grössen verwendet.

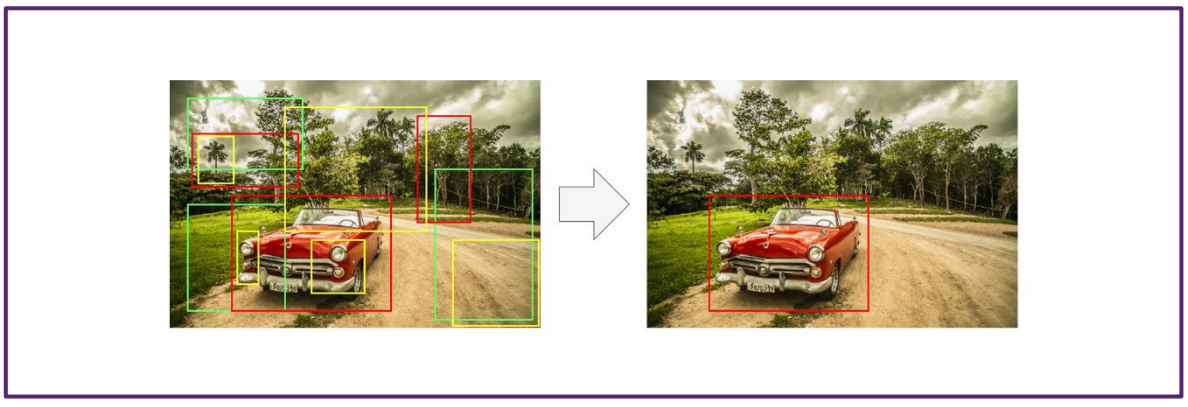


Abbildung 23: Object Detection: Anchor Box – Merge (Bild Oldtimer: [8])

7.1.3 Anchor Box Ebene

Mit der Hilfe dieser Boxen wird die Geschwindigkeit und die Effizienz des Erkennungsteils erhöht. Anchor-Boxen sind vordefinierte Bounding Boxen mit einer bestimmten Höhe und Breite. Die Trainingsdatensätze und die sich darin befindenden Objektgrössen werden verwendet, um Boxen zu definieren, welche den Massstab und das Seitenverhältnis bestimmter Objektklassen erfassen. Während der Objekterkennung werden die Anchor Boxen über das Bild gekachelt. Für die Anchor Boxen werden dann vom Netzwerk die Wahrscheinlichkeit und weitere Attribute wie beispielsweise der IoU und Offset für jede Box prognostiziert. Diese Vorhersagen werden dann verwendet, um die einzelnen Boxen zu präzisieren. [11]

7.1.4 Klassifizierung und Regression

Die Anchor Box Objekte werden mit einem Klassifizierungs- und einem Regressionszweig verbunden. Der Klassifizierungszweig hat Faltungsschichten, welche die Klasse für jede Anchor Box vorhersagen. Der Regressionszweig hat diese Schichten für die Vorhersage der Offset. Die letzte Convolution-Schicht vor der Merge-Schicht muss dann die passende Anzahl Filter haben:

- Klassifizierung: Anzahl Anchor Boxen + 1 (für die Hintergrunds Klasse)
- Regression: Viermal die Anzahl der Anchor Boxen.

7.1.5 Merge

Die Outputs der Klassifizierungszweige werden mittels Merge-Funktionen zusammengeführt. Die Outputs der Regressionszweige werden mit Hilfe der RCNN Box Regression kombiniert. [12] [13] [10] [14]

7.2 Neuronales Netzwerk

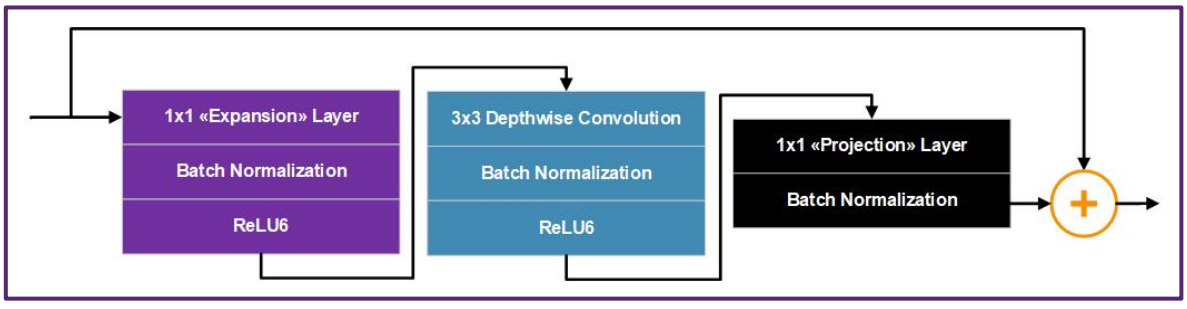


Abbildung 24: Object Detection: MobileNetV2 – Baustein

Das eigentliche neuronale Netzwerk, auf welchem SSD läuft, ist das MobileNetV2. Es verwendet «Depthwise Separable Convolutions», was es ihm ermöglicht, ca. 9-mal weniger Arbeit wie vergleichbare neuronale Netzwerke leisten zu müssen. Mit der Nutzung von «Residual Connection» ist es möglich, den Gradienten direkt durch das Netzwerk fliessen zu lassen, ohne dabei nicht-lineare Aktivierungsfunktionen zu durchlaufen. Dies optimiert die Geschwindigkeit des Netzwerks und führt schlussendlich,

wie aus der Evaluation der letzten publizierten Arbeit hervorging, zudem zu einem geringeren Batterieverbrauch. [1] [15]

In der Abbildung 24 wird ein Baustein des MobileNetV2 dargestellt, welcher aus drei «Depthwise Separable Convolution» Schichten besteht.

In der ersten Schicht kommt zuerst eine 1x1 Faltung, welche die Anzahl der Kanäle in den Daten erweitert, bevor sie in die «Depthwise Convolution» gehen. Daraus lässt sich schliessen, dass diese «Expansion» Schicht immer mehr Aus- wie Eingangskanäle hat. Die genaue Ausdehnung der Daten wird mit dem Ausdehnungsfaktor definiert. Danach kommt eine «Batch Normalization»: diese standardisiert die Eingaben für die Schicht und stabilisiert damit den Lernprozess, worauf die Anzahl der benötigten Trainingsepochen drastisch reduziert wird. [16]

Am Ende der Schicht kommt die Aktivierungsfunktion. MobileNetV2 verwendet ReLU6, welche verhindert, dass die Aktivierungen zu gross werden, und lässt die Form der Funktion eher wie eine Sigmoid-Funktion aussehen. ReLU6 ist robuster als das reguläre ReLU bei der Verwendung von Berechnungen mit geringer Genauigkeit (bezogen auf Festkomma-Arithmetik). [17]

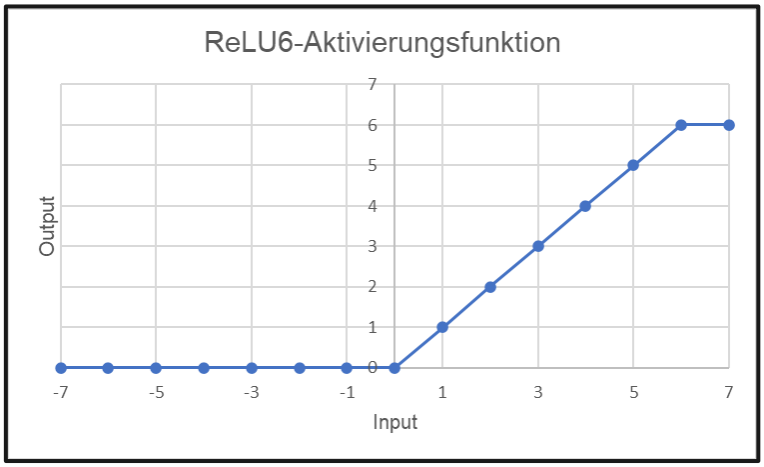


Abbildung 25: Object Detection: ReLU6

In der zweiten Schicht kommt die «3x3 Depthwise Convolution». Bei dieser werden die Eingangskanäle nicht wie bei einer regulären Faltung kombiniert, sondern die Faltung wird für jeden Kanal separat durchgeführt. Für ein Bild mit drei Kanälen erzeugt eine «Depthwise Convolution» ein Output-Bild, das ebenfalls drei Kanäle hat. Ein Gewichtungssatz wird dann für jeden Kanal vergeben. Der eigentliche Zweck der «Depthwise Convolution» ist die Filterung der Eingangskanäle[3]. Danach kommt wieder eine «Batch Normalization» und die ReLU6-Aktivierungsfunktion. [18]

[3] Man denke zum Beispiel an die Kantenerkennung oder Farbfilterung etc.

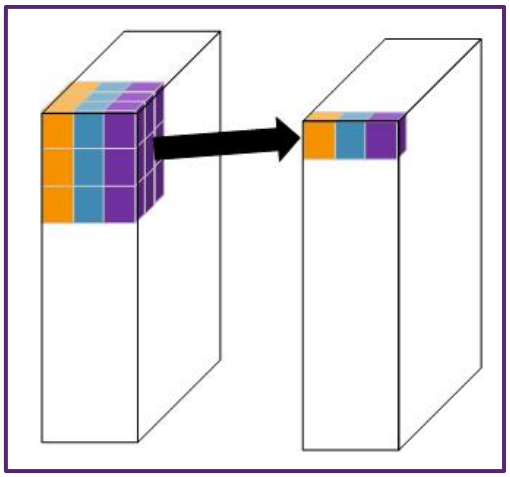


Abbildung 26: Object Detection: Depthwise Convolution

In der letzten Schicht, dem «1x1 Projection Layer», werden die Anzahl Kanäle reduziert. Darum nennt man sie Projektionsschicht. Die Daten, mit einer hohen Anzahl Kanälen (Dimensionen), werden in einen Tensor mit einer viel geringeren Anzahl an Kanälen projiziert.

Das MobileNetV2 unterscheidet sich hierdurch auch zu traditionellen Residualmodellen, denn der Ein- und Ausgang des ganzen Bausteins bilden dünne Bottleneck-Schichten. Dadurch ist es eine invertierte Residualstruktur.

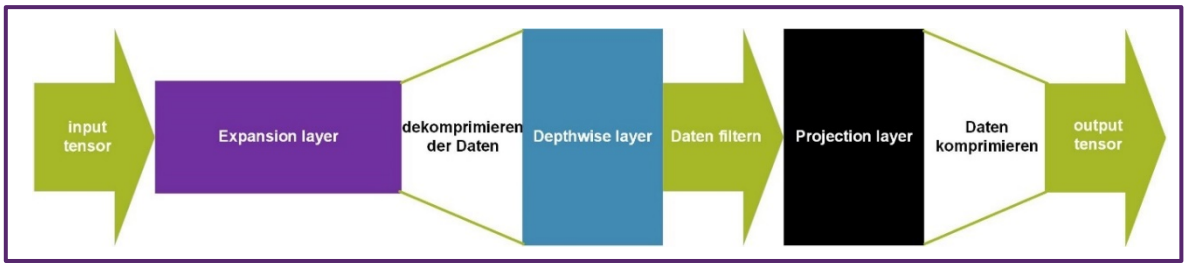


Abbildung 27: Object Detection: MobileNetV2 Prozess

Die ganze Architektur vom MobileNetV2 besteht aus 17 aneinandergereihten Bausteinen. Darauf folgt eine reguläre 1x1-Faltung, dann eine globale Durchschnitts-Pooling-Schicht, um eine Feature-Map für jede entsprechende Kategorie der Klassifikationsaufgabe zu erzeugen, und am Ende noch eine Klassifikationsschicht. [19]

Dies hat den grossen Vorteil, dass die Expansion und die Projektion mit Hilfe von Faltungsschichten gemacht werden, welche lernbare Parameter haben. So kann das Modell lernen, wie es die Daten an jeder Stelle des Netzwerks am besten de/komprimieren soll. [20] [21] [22]

7.3 Training Set

Das Modell ist mit COCO (Common Objects in Context) trainiert. COCO ist ein umfangreicher Datensatz zur Objekterkennung, Segmentierung und Beschriftung. Grosse Sponsoren dieses Training Sets sind unter Anderem Microsoft, Facebook und CVDF. Das Datenset besteht aus 330'000 Bildern, davon mehr

als 200'000 gelabelt. Das Datenset umfasst aktuell 1.5 Millionen Objektinstanzen. Jedes Bild enthält dazu fünf Beschriftungen. Es wird in 80 «Thing»-Kategorien (z.B. Person, Auto, Banane) und einer Teilmenge aus 91 «Stuff»-Kategorien (Strasse, Himmel, Gras) unterschieden. [23] [24]

8. Collision Avoidance

Die Collision Avoidance wird verwendet, um eine Kollision mit dem Objekt beim direkten Anfahren zu verhindern und um Kanten (beispielsweise Tischkanten) zu erkennen. Für die Implementation wird das AlexNet verwendet, welches sich im Vergleich zu anderen Netzen in den Punkten Trainingszeit und Speicherverbrauch bewährt, was für den Roboter mit beschränkten Ressourcen sehr vorteilhaft ist. In den Tests konnte es zusätzlich mit seiner Genauigkeit überzeugen. Die Implementation des Modells und des neuronalen Netzwerks geschieht über das PyTorch Framework. Dies ist ein Open Source Machine Learning Framework. Nach der Initialisierung wird auch hier CUDA und somit die GPU-Rechenleistung verwendet. [25]

8.1 Modell und Training Set

Die Collision Avoidance bietet die Möglichkeit, ein Modell selbst auf der Basis von einem eigenen Training Set zu trainieren. Hier wird ein vortrainiertes Modell von NVIDIA JetBot verwendet. Das Modell ist mit einem begrenzten Datensatz mit einer Raspberry Pi V2 Kamera mit Weitwinkelvorsatz trainiert. Es hat sich in den Testläufen als sehr stabil und genau erwiesen.

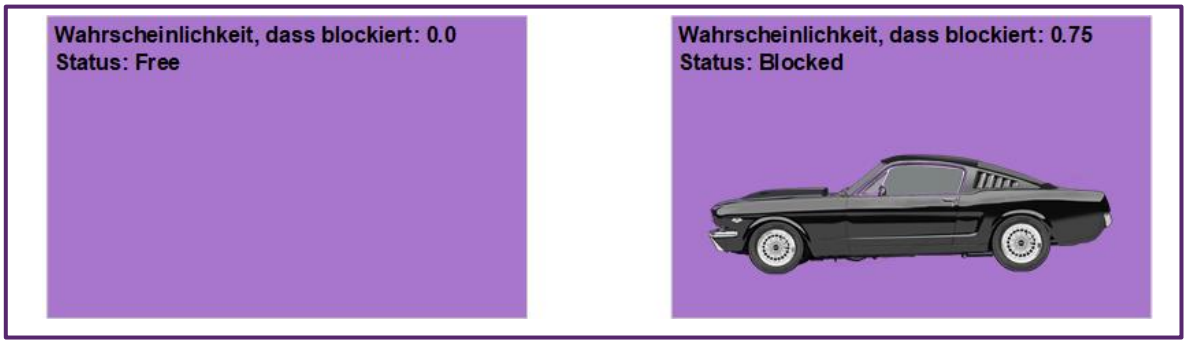


Abbildung 28: Collision Avoidance: Modell – Free / Bocked (Bild Ford Mustang: [26])

Das Modell gibt zwei Zustände zurück; «Free» oder «Blocked». Somit kann eine Wahrscheinlichkeit berechnet werden, ob es frei oder eben blockiert ist. Es wird vor dem Gebrauch des Modells definiert, bei welcher Wahrscheinlichkeit das Modell den Status «Blocked» zurückgeben soll. Dies bietet eine individuelle Anpassungsmöglichkeit je nach Grösse der Objekte. [27] [28]

8.2 Neuronales Netzwerk

Als neuronales Netzwerk wird AlexNet verwendet, welches 60 Millionen Parameter und 650’000 Neuronen hat. Es ist die erste gross angelegte faltungsneuronale Netzwerkarchitektur, welche bei der Klassifikation von ImageNet gut abschneidet. ImageNet ist eine Bilderdatenbank für Deep Learning, welche aus über einer Million Trainingsbildern, 50'000 Validierungsbildern und 100'000 Testbildern besteht. [29] Die Architektur des AlexNets besteht aus fünf Faltungsschichten und drei vollständig verbundenen Schichten. Als Aktivierungsfunktion wird zum ersten Mal in einer Architektur ReLU nicht-Linearität verwendet. Diese gibt die Eingabe direkt aus, wenn sie positiv ist, andernfalls gibt sie Null zurück.

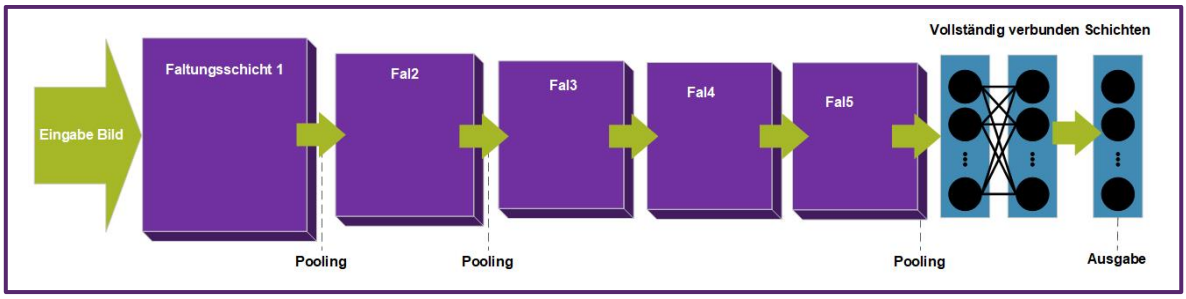


Abbildung 29: Collision Avoidance: AlexNet – Architektur

Als Input erhält das AlexNet ein RGB-Bild mit einer Grösse von 256x256 Pixel. Hat das Bild eine andere Grösse, so muss zuerst eine Grössenänderung vorgenommen werden. Aus diesem Bild werden zufällige Ausschnitte à 227x227 genommen, um die erste Faltungsschicht zu füttern.

Mit mehreren Faltungskernen (Filtern) werden aus dem Bild interessante Merkmale extrahiert. Gibt es in einer Faltungsschicht mehrere Kernels der gleichen Grösse, so hat die erste Faltungsschicht beispielsweise 96 Kernel mit einer Grösse von 11x11x3 (Höhe x Breite x Kanäle).

Wie auf der Abbildung zu sehen ist, folgen den ersten beiden und der fünften Faltungsschicht ein überlappende Max-Pooling-Schicht. Diese sind Maxpool-Schichten, deren Schrittweite kleiner als die Fenstergrösse sind. Beim Max-Pooling wird das Bild entlang seiner räumlichen Dimensionen für die nächste Schicht heruntergesampelt. Die Schichten drei bis fünf sind direkt miteinander verbunden. [30] [31]

Nach der fünften Faltungsschicht und der überlappenden Max-Pooling-Schicht folgen zwei vollständig verbundene Schichten. Bei dieser neuronalen Netzstruktur sind alle Neuronen mit allen Inputs und Outputs verbunden. Die Objektinformationen werden in diese beiden Schichten eingespeist und die zweite Schicht speist in den «Softmax Classifier» mit 1000 Klassenlabels ein. Dieser gibt als Output eine Wahrscheinlichkeit der Klassenzugehörigkeit für jedes Klassenlabel aus. [32]

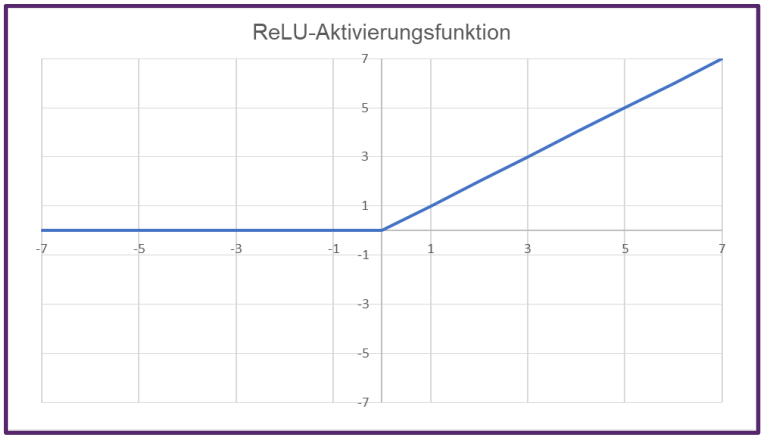


Abbildung 30: Collision Avoidance: ReLU

Die ReLU Nicht-Linearität Aktivierungsfunktion wird nach allen Faltungs- und vollverknüpften Schichten angewendet, um die Ergebnisse der Schichten zu aktivieren. Die Funktion sorgt dafür, dass alle Werte,

die kleiner als Null sind, im Output zu Null werden und alle, die grösser als Null sind, werden gleichbleibend weitergegeben. [33] [34] [35] [36] [37] [38] [39]

8.3 Direktes Anfahren des Objekts

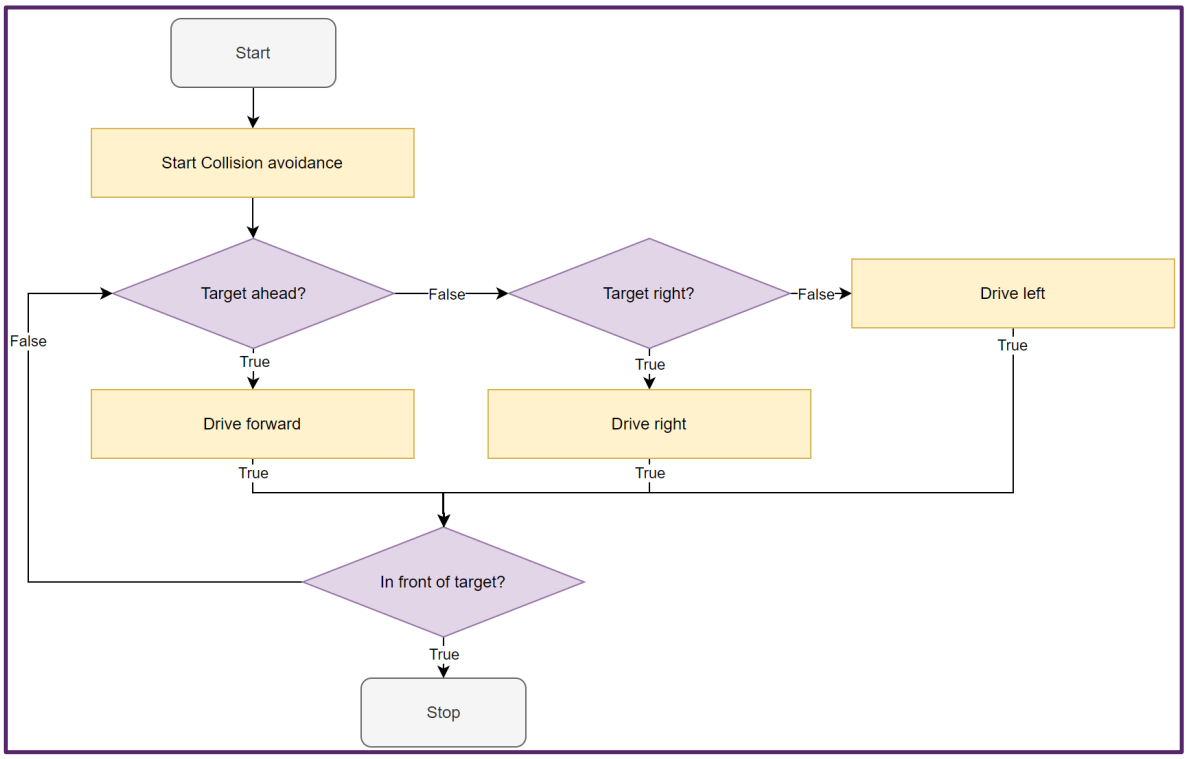


Abbildung 31: Collision Avoidance: Direktes Anfahren

Sobald das gewählte Objekt erkannt wurde, startet die Collision Avoidance und der Algorithmus für das direkte Anfahren eines Objekts. Der Algorithmus verwendet die Object Detection, um herauszufinden, wie sich der Roboter ausrichten muss.

In einem ersten Schritt dreht sich der Roboter in Richtung Objekt, bis sich dieses innerhalb eines definierten Offsets zur Mitte der Kamera befindet. Im zweiten Schritt fährt der Roboter geradeaus auf das Objekt zu. Dies kann dazu führen, dass das Objekt wieder ausserhalb des Offsets zur Mitte der Kamera landet. In diesem Fall wird wieder der erste Schritt, gefolgt vom zweiten ausgeführt.

Sobald die Collision Avoidance eine Kollision detektiert, wird der Roboter gestoppt.

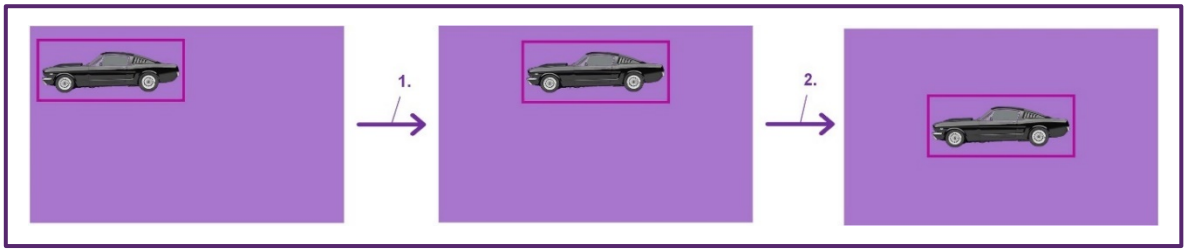


Abbildung 32: Collision Avoidance: Direktes Anfahren – Beispiel (Bild Ford Mustang: [26])

Ausgangslage: Es wird ein Auto links oben im Bild erkannt.

1: Der Roboter richtet sich nach links aus.

2: Der Roboter fährt auf das Auto zu und die Collision Avoidance alarmiert. → Der Roboter hält an. → Der Roboter befindet sich nun direkt vor dem Auto.

8.4 Kantenerkennung

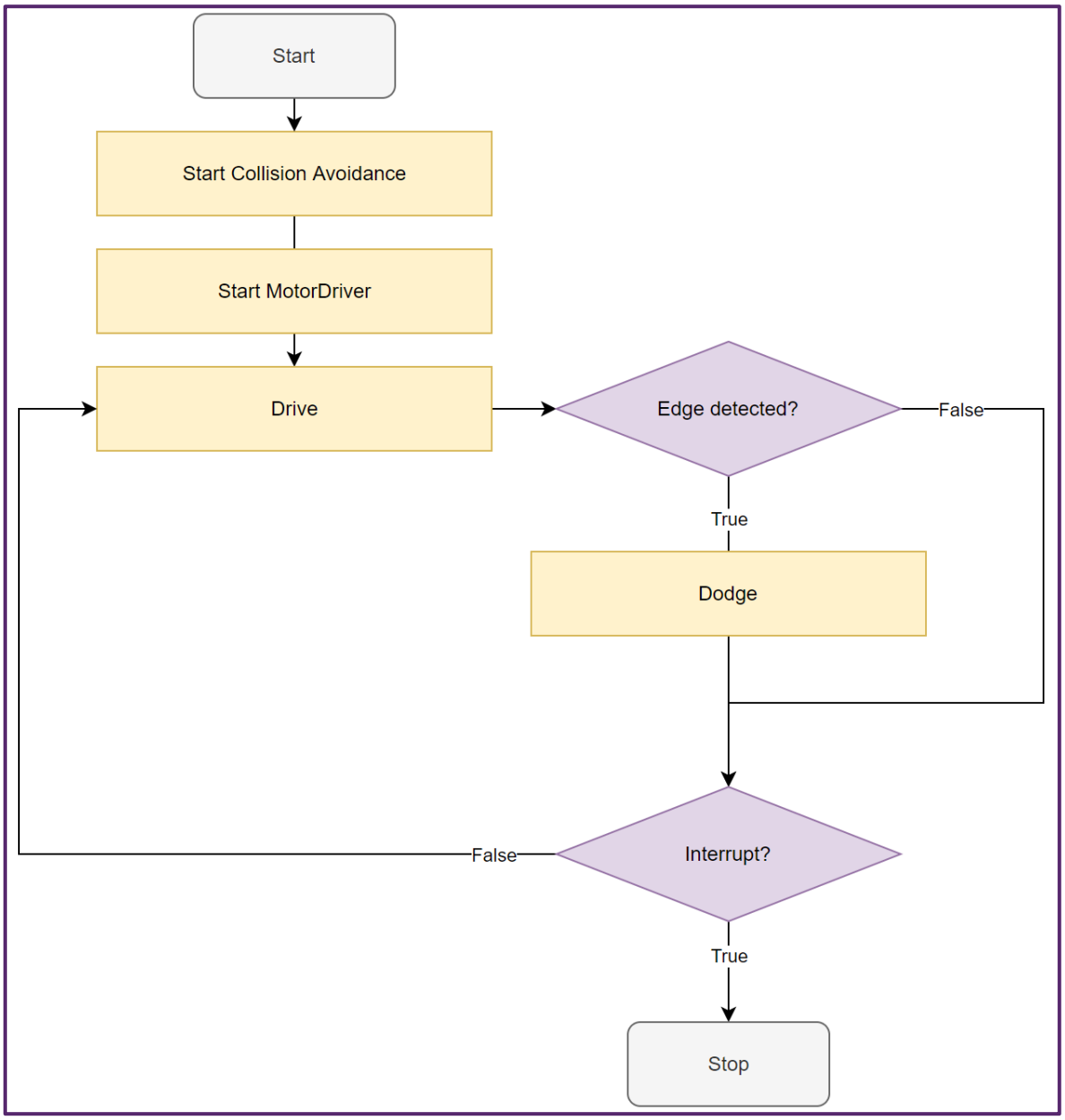


Abbildung 33: Collision Avoidance: Kantenerkennung

Die Kantenerkennung ist ein eigenständiges Programm, welches die Collision Avoidance nutzt. Das verwendete Modell ist selbst mit einem Datensatz aus eigenen Bildern trainiert, welche mit dem JetBot gemacht wurden. Es wurden Fotos von Kanten und auch Gegenständen gemacht, welche der Roboter

als Kollision erkennen soll. Zudem wurden auch Bilder gemacht, bei welchen der Roboter im freien Zustand ist. Der Roboter wurde auf einem hellen so wie auf einem dunklen Untergrund trainiert, da sich während den Tests gezeigt hat, dass der Roboter viel schlechter Kanten detektiert, wenn dies nicht gemacht wird.

Sobald das Modell und die Motoren geladen sind, läuft die Applikation in einer Dauerschleife. Der Roboter fährt und verarbeitet gleichzeitig die Kameraaufnahmen. Solange keine Kante erkannt wird, fährt der Roboter gerade aus. Wird eine Kante erkannt, wendet er sich von dieser ab und fährt weiter. Die Applikation kann jeder Zeit mit einem Interrupt beendet werden.

9. Testing Roboter

In diesem Kapitel wird die Vorgehensweise beschrieben, wie der Roboter getestet wird.

9.1 Embedded Testing

Da der Roboter sehr modular aufgebaut ist und viele Komponenten extern an das Nvidia Entwicklerboard angeschlossen wurden, sind automatisierte «Embedded Tests» vorhanden. Zusätzlich zu den Hardwarekomponenten werden aber auch essenzielle Software-Module getestet. Diese Tests sind mit dem Framework «unittest» und Python3 implementiert. Die folgenden Komponenten werden mit diesen Tests abgedeckt:

Tabelle 12: Testing Roboter: Embedded Testing Komponenten

Komponente	**Beschreibung**
Arduino Mikrokontroller	Kann der Kontroller angesteuert werden?
Ultraschallsensoren (3x)	Geben die Sensoren mögliche reale Werte zurück?
Kamera	Kann die Kamera angesteuert werden?
Motor Controller, Motoren und Display	Können die Komponenten über die I^2C Schnittstelle angesteuert werden?
WLAN	Kann über das WLAN Modul eine Verbindung ins Internet aufgebaut werden?
Object Detection Modell	Ist das Modell zur Object Detection vorhanden?
Collision Avoidance Modell	Ist das Modell zur Collision Avoidance vorhanden?
Collision Avoidance Modell (Edge Detection)	Ist das Modell zur Collision Avoidance während der Edge Detection Applikation vorhanden?
Custom Algorithm Library	Ist die Library zur Problemlösung vorhanden?

Das Testprotokoll mit genaueren Angaben (Test-id, Action, Setpoint und Status) zu den einzelnen Tests ist im Anhang zu finden.

9.2 Trial-and-Error Testing

Da viele Algorithmen nur nach Beobachtung in der Testumgebung evaluiert werden können, sind viele Anpassungen nach der Trial-and-Error Methode verbessert worden. Die Testumgebung ermöglicht es aber, die einzelnen Runs mit den gleichen Bedingungen durchzuführen. Mit dieser Testmethode sind hauptsächlich Parameter zur algorithmischen Pfadfindung verbessert worden.

Für das Testen der Verhaltensweise des Roboters in realen Bedingungen sind keine automatisierten Tests möglich. Viele Verbesserungen sind anhand von menschlichen Beobachtungen vorgenommen worden.

10. Web-App

Bereits in der Studienarbeit vom Herbstsemester 2020 wurde ein kleines Web-App implementiert. Dieses diente zur Visualisierung der Pfadfindung sowie zur kurzen Erläuterung der Arbeit.

Für die Bachelorarbeit wurde das Backend der Visualisierung der Pfadfindung übernommen, dann aber durch viele visuelle Anpassungen verbessert und weitere neue Features, wie die Anbindung des Roboters, hinzugefügt und durch eine Erläuterung der Theorie ergänzt. Ein visueller Vergleich der zwei Versionen ist im Anhang zu finden.

Die Web-App soll dazu genutzt werden, einen tieferen Einblick in die Theorie zu erhalten. Zudem können Teilnehmer einer Informationsveranstaltung detailliertere Informationen zum Roboter und seiner erledigten Arbeit erfahren.

Abbildung 34: Web-App: QR-Code zur Web-App https://pt-env-applied-ml.azurewebsites.net/

10.1 Architektur

Das Web-App läuft auf einem Azure App Service (F1 Free Plan) und wird automatisch bei einem «push» auf den Master-Branch des Repository «ba-fs2021-web-app» deployed.

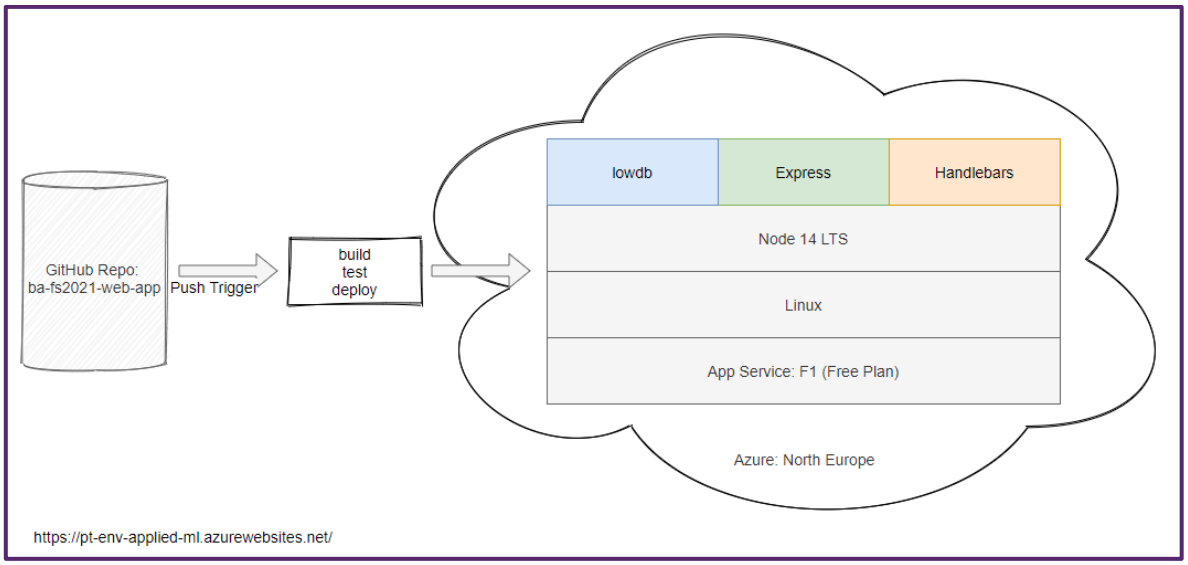


Abbildung 35: Web-App: GitHub - Azure Integration

10.1.1 Technology Stack

Das Web-App hat folgende Abhängigkeiten:

- lowdb
  - https://www.npmjs.com/package/lowdb
  - Kleine lokale JSON Datenbank
  - Auf der Datenbank Instanz werden Log-Files vom physischen Roboter gespeichert
- express
  - https://www.npmjs.com/package/express
  - Minimalistisches Web Framework
- express-handlebars
  - https://www.npmjs.com/package/express-handlebars
  - View Engine
- dateformat
  - https://www.npmjs.com/package/dateformat
  - Framework für besseres Handling von Zeit und Datum
- bcyrpt
  - https://www.npmjs.com/package/bcrypt
  - Password Hashing Library
  - Wird verwendet, um Datenbank vor ungewollten Schreibzugriffen zu schützen
- mocha
  - https://www.npmjs.com/package/mocha

- Test-Framework

- chai
  - https://www.npmjs.com/package/chai
  - Assertion library

10.1.2 Kommunikation Roboter und Datenbank

Damit der Roboter Informationen von physischen Ausführungen dem Web-App zur Verfügung stellen kann, wurde ein Zugang für externe Geräte ermöglicht.

Für die Datenbank wurde folgende Funktionalität implementiert:

Tabelle 13: Web-App: Datenbank Funktionalität

Funktion:	**Beschreibung:**	**Für externe Nutzung:**
addLog(content)	- Wird genutzt, um neue Logs der Datenbank hinzuzufügen - Gesicherte Route	Ja
getAll()	- Gibt alle Datenbank Einträge zurück	Nein
reset()	- Löscht alle Daten in der Datenbank und stellt Originalschema her	Nein

Damit der Roboter über die Funktion «addLog(content)» neue Logs erstellen kann wurde eine gesicherte POST-Route erstellt.

```
1 const express = require('express');
2 const router = express.Router();
3 const controller = require('../controller/controller');
4
5 router.get('/', controller.simulation);
6 router.get('/readme', controller.readme);
7 router.get('/remote', controller.remote);
8
9 router.post('/addToDb', controller.addToDb);
10
11 module.exports = router;
```

Abbildung 36: Web-App: Router (index.js), Line 9: POST-Route

```
17 function addToDb(req, res){
18     let password = req.body.password;
19     let hash = '[redacted].';
20
21     let contentToAdd = String(req.body.content);
22
23     bcrypt.compare(password, hash, function(err, result) {
24         if(result){
25             dbService.addLog(contentToAdd);
26         }
27     });
28
29     res.redirect('/');
30 }
```

Abbildung 37: Web-App: Controller (controller.js), addToDb-Ausschnitt

Bei einem POST-Request auf «https://pt-env-applied-ml.azurewebsites.net/addToDb» wird aus dem Body des Requests das mitgegebene Passwort mit dem Hash des «API-Key» verglichen. Dies wird mithilfe von dem Bcyrpt-Algorithmus realisiert. Der API-Key besitzt folgende Eigenschaften:

- Bcyrpt-Hash
  - 10 Salt-Rounds

Wenn das übergebene Passwort mit dem Hash übereinstimmt, wird ein neuer Log-Entry mit dem Inhalt aus dem Body in die Datenbank geschrieben. Die Kommunikation mit dem Web-App erfolgt ausschliesslich über https-only und einer minimalen TLS Version von 1.2.

10.2 Features

Die folgende Tabelle dient als Übersicht, welche Features das Web-App unterstützt:

Tabelle 14: Web-App: Features

Name:	URL:	Beschreibung:
#SIMULATION	/	Auf dieser Sub-Page können Pfadfindungsprobleme simuliert und der Lösungsansatz visualisiert werden. Das Ziel dieser Seite ist es, die algorithmische Pfadfindung besser zu verstehen.
#REMOTE	/remote	Auf der «Remote» Sub-Page können Detailinformationen von realen physischen JetBot Ausführungen angezeigt werden. Diese Log-Files werden nach Datum und Zeit gruppiert.

		Das Ziel dieser Seite ist es, den Ablauf des Roboters besser zu verstehen.
#README	/readme	Die «Readme» Sub-Page gibt einen Einblick in die Arbeit und zeigt die wichtigsten Entscheidungen und theoretischen Grundlagen auf. Das Ziel dieser Seite ist es, den Aufbau und die Theorie hinter dieser Bachelorarbeit besser zu verstehen.

10.3 Testing

Das Testen der Web-App wurde in zwei Teile unterteilt.

10.3.1 Web-App

Die Web-App und all ihre Sub-Pages wurden mithilfe des Open-Source Tools «Google Lighthouse» getestet. Die folgenden Kategorien wurden jeweils für Mobile- und Desktop-Endgeräte getestet:

- Performance (Durchschnittlich erreichte Punkte über alle Pages: 95.00)
- Accessibility (Durchschnittlich erreichte Punkte über alle Pages: 69.00)
- Best Practices (Durchschnittlich erreichte Punkte über alle Pages: 93.00)

Das detaillierte Testprotokoll ist im Anhang vorhanden. Die Bewertung wurde mit einer Skala von 0-100 Punkte realisiert. Ein Wert zwischen 0 – 55 wurde als schlecht, ein Wert zwischen 56 – 85 als genügend bis gut und ein Wert zwischen 86 – 100 als sehr gut klassifiziert.

10.3.2 Database

Die Datenbank Funktionalität wurde mithilfe des Test-Runners «mocha» und der Assertion Library «chai» getestet. Folgende Unittests wurden durchgeführt:

Tabelle 15: Web-App: «local database tests»

local database tests	
reset test	- Die Datenbank wird zurückgesetzt und anschliessend geprüft, ob «getAll().length» gleich 0 ist
addLog test	- Es wird ein neuer Test Log hinzugefügt und anschliessend geprüft, ob der Inhalt des Logs in der Datenbank vorkommt
getAll test	- Es wird geprüft ob «getAll()» ungleich 0 ist
Nach allen Tests wird die Datenbank zurückgesetzt.	

Ein genaueres Testprotokoll ist im Anhang zu finden.

11. Schlussfolgerung

In dieser Arbeit konnten in einem ersten Schritt die Möglichkeiten und Grenzen des JetBots anhand von Beispielsapplikationen erforscht, getestet und aufgezeigt werden.

Mit den Applikationen wird der Roboter befähigt, sich in Räumen algorithmisch zu bewegen sowie Objekte und Kollisionen mit Hilfe von künstlicher Intelligenz zu erkennen.

Aus den Performancetests geht hervor, dass der Roboter vor allem bei der RAM-Auslastung an seine Grenzen stösst. Jedoch lassen auch fortwährend Peaks bei der GPU-Auslastung vermuten, dass wenn man ein weiteres drittes Modell auf dem Roboter laufen lassen würde, dieser überlastet wäre.

In Bezug auf die Möglichkeiten werden im Ausblick weitere Beispiele erläutert.

11.1 Ausblick

Durch den modularen Aufbau des Nvidia Developer-Kits gibt es unzählige Funktionen, die in der Zukunft noch implementiert werden können. Der grösste limitierende Faktor ist hierbei die Performance der Hardware und insbesondere die kleine Kapazität des Arbeitsspeichers. Trotzdem wurden während dem Arbeiten mit dem Roboter einige zusätzliche Ideen für potenzielle Implementation gefunden, welche ein Teil einer zukünftigen Arbeit sein könnten. Diese werden nachfolgend kurz erläutert.

11.1.1 Fortlaufendes Lernen

Eine Implementation eines ML-Modells, welches fortlaufend erweitert wird. Diese Implementation könnte entweder auf eine neue Aufgabenstellung angewendet werden oder auf ein Modell, das bereits in dieser Arbeit verwendet wird, wie zum Beispiel die Collision Avoidance.

Bei der Collision Avoidance gibt es Grenzfälle, die schwierig einzuschätzen sind, wodurch der Collision-Faktor tief bleibt, obwohl der Roboter sich trotzdem kurz vor einer Kollision befindet. Bei diesen Grenzfällen wäre ein Ansatz, dass der Roboter solche Situationen in das bereits bestehende Modell integriert und so dazulernt. Um diese Situationen zu erkennen, könnte man das externe Feedback von einem Operator miteinbeziehen oder sich auf andere Sensoren verlassen, die dann als Feedback-Quelle dienen.

Bei dieser Erweiterung muss man aber auf das bekannte Machine Learning Problem «Catastrophic Forgetting» Rücksicht nehmen. Bei dieser Problematik werden zuvor gelernte Informationen durch die neuen Informationen teilweise vergessen oder überschrieben [40].

11.1.2 Sprachsteuerung

Eine Sprachsteuerung für den Roboter wäre interessant, da die Steuerung und Interaktion mit dem JetBot momentan nur über den Jupyter Notebook Server läuft, was ein Client-Device voraussetzt.

Würde man diesen Use-Case implementieren, könnten folgende Beispielsbefehle durchaus Sinn ergeben:

- «JetBot, suche nach einem Auto!»
- «JetBot, bewege dich selbständig auf dem Tisch!»

Falls die Sprachsteuerung auf dem Roboter selbst implementiert wird, muss die Performance Evaluation miteinbezogen werden, da die Ressourcen mit den aktuellen Modellen bereits stark ausgelastet sind.

11.1.3 Interkommunikation

Suchprobleme könnten durch einen zweiten Roboter schneller gelöst werden. Würde man beispielsweise einen zweiten Roboter gleichzeitig an einem anderen Ort im Modell starten, könnten diese gemeinsam nach dem Ziel suchen. Um diese Aufgabe performant zu lösen, wäre ein Kommunikations-Channel zwischen den beiden Roboter sinnvoll.

Nach dem Aufbau eines solchen Channels könnten zusätzlich auch Trainingsdaten oder komplette Modelle ausgetauscht werden.

11.1.4 Raum Mapping

Da das Entwicklerboard durch weitere Sensoren erweitert werden kann, wäre eine Raum Mapping Anwendung sehr interessant. Durch eine Implementation könnte der Roboter nach dem Erkunden der Räumlichkeiten eine Map erstellen. Mit diesen Informationen könnte dann mithilfe eines klassischen Pathfinding-Algorithmus, wie zum Beispiel der Dijkstra Algorithmus, der schnellste Weg vom Startpunkt zum Ziel berechnet werden. Diese shortest Path Berechnung wurde bereits als Beispielanwendung in der Web-App unter dem Tab Simulation (https://pt-env-applied-ml.azurewebsites.net/) implementiert.

11.1.5 Erweiterung der Anbindung des Web-Apps

Die Anbindung des Web-Apps könnte durch neue Funktionen erweitert werden. Die folgende Liste enthält Funktionen, die das Nutzungserlebnis verbessern würden:

- Befehle an den Roboter direkt über das Web-App senden
- Live-Stream der Kamera und Screenshot, wenn der Roboter das Ziel gefunden hat mit Informationen zur Object Detection und Collision Avoidance
- Visualisierung der Räume, zum Beispiel als 2D Karte

Bei regelmässiger Nutzung des Web-Apps wäre zudem ein Upgrade auf einen Paid-Azure-Plan sinnvoll, da beim Free-Plan keine SLAs gelten und die Ressourcen geteilt werden [41].

11.1.6 Performance Optimierungen und Hardwareanpassungen

Durch die eigenschränkten Hardwareeigenschaften stösst man schnell an die Kapazitätsgrenzen des Roboters. Um diese Limitierung anzuheben sind zwei Ansätze erfolgsversprechend: Zum einen könnte

ein zweites Entwicklerboard miteingebaut werden, zum anderen könnten Cloud-Dienste verwendet werden.

Ansatz 1: Zweites Entwicklerboard

Bei dieser Variante könnte dem Roboter ein zweites Entwicklerboard hinzugefügt werden. Dadurch könnte die parallele Ausführung von mehreren Modellen verteilt und so die Hardware-Limitierung angehoben werden. Die folgende Abbildung zeigt ein selbst erstellter Prototyp (https://github.com/yV11/ba-fs2021/blob/main/3d-models/chassi-experimental.stl) eines 3D Modells, welches eine zweite Ebene für das zusätzliche Nvidia-Board besitzt.

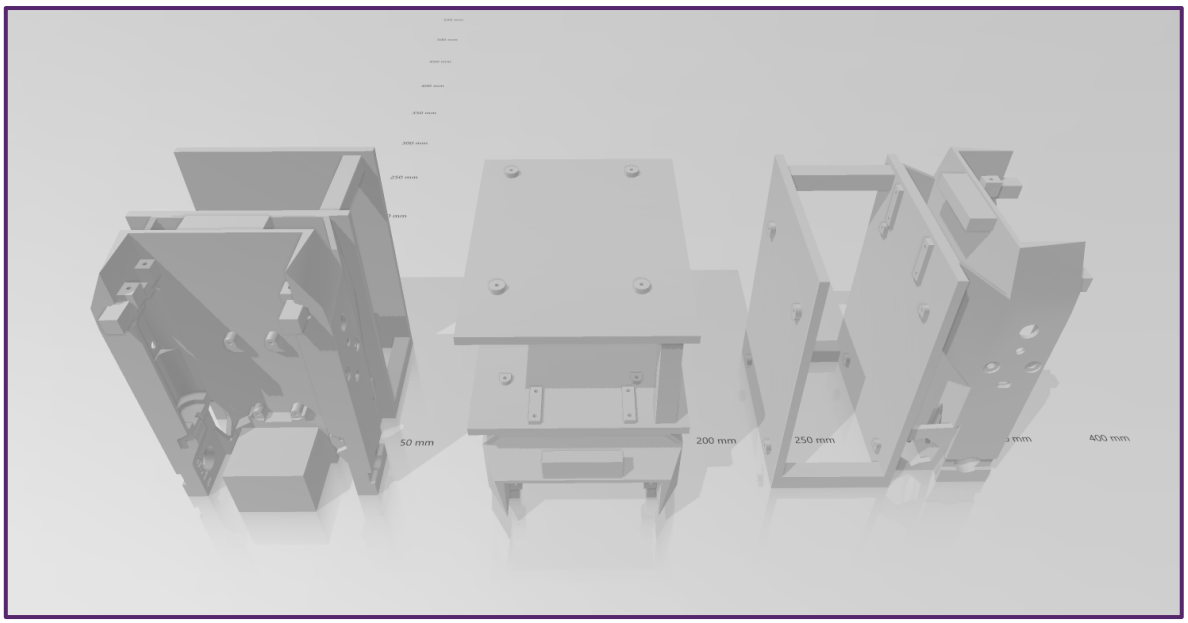


Abbildung 38: Schlussfolgerung: 3D-Modell Erweiterung, Prototyp

Für diese erweiterte Variante müssten die Motoren sicher stärker angesteuert werden, da das Gewicht deutlich höher wäre. Ein weiterer Punkt, der genau analysiert werden müsste, ist die Stromversorgung. Das zusätzliche Board bräuchte wahrscheinlich eine zweite eigene Powerbank. Die Kommunikation zwischen den zwei Boards müsste auch selbst implementiert werden, damit eine Art Cluster-Computer entsteht.

Ansatz 2: Cloud Computing

Die Hardware-Limitierung könnte auch durch das Auslagern von KI-Modellen in die Cloud erhöht werden. Dieser Ansatz wurde nicht genauer analysiert, folgende Punkte müssten aber sicher beachtet werden (nicht abschliessend):

- Kosten des Cloud-Anbieters
- Delay zwischen Anfrage und Antwort (Response-Time)
- Anforderung an die Internetverbindung (Übertragungsgeschwindigkeit)

Mögliche Anbieter wären die Folgenden (nicht abschliessend):

- Microsoft Azure (https://azure.microsoft.com/en-us/)
- Amazon (AWS) – Cloud Computing Services (https://aws.amazon.com/)
- Google Cloud Platform (https://cloud.google.com/)

12. Danksagung

Wir möchten uns bei all denjenigen bedanken, welche uns während des Erarbeitungsprozess dieser Bachelorarbeit und generell während des Studiums unterstützt haben.

Ein besonderer Dank geht an Herr Prof. Dr.-Ing. Andreas Rinkel und Herr Marc Sommerhalder, welche unsere Arbeit betreut und uns fachlich unterstützt haben.

Letztlich möchten wir uns bei Nadja Rek und Ronald Peter für das Korrekturlesen unserer Arbeit und für ihre konstruktive Kritik bedanken.

13. Glossar

C

CNN 32, *Convolutional Neural Network*

CSV 20, *Dateiformat: Comma Seperated Values*

CUDA 20, 32, 38, *Compute Unified Device Architecture*

CVDF 36, *Common Visual Data Foundation*

F

faltungsneuronale Netzwerkarchitektur 38, *Netzwerk bestehend aus mehreren Schichten künstlicher Neuronen*

Faltungsschichten 34, 36, 38, *Implementiert unterschiedliche Filter und integriert diese in das neuronalen Netzwerk*

Feature-Maps 32, 33, *Ausgabe eines Filters, der auf die vorherige Schicht angewendet wurde*

G

GPIOs 13, *General Purpose Input/Output*

I

IoU 34, *Intersection over Union*

M

Mikrokontroller 13, 16, 17, 18, 43, *Halbleiterchips, die einen Prozessor und zugleich auch Peripheriefunktionen enthalten*

ML 8, 11, 20, 22, 23, 26, 50, *Machine Learning*

MultiBox 32, *Name des Verfahrens zur Bounding-Box-Regression*

N

NVIDIA TensorRT SDK 9, 32, *Hochleistungsfähige Deep-Learning-Inferenz aufgebaut auf CUDA*

O

Offset 34, *Versatz*

Offsets 40, *Versatz*

OLED 14, *Organische Leuchtdiode (organic light emitting diode)*

P

Pinouts 12, *Pinbelegung*

PLA 15, *Polylactid, Material für 3D Druck*

PyTorch 9, 38, *Open Source Machine Learning Framework*

R

RCNN 34, *Region Convolutional Neural Network*

ReLU 35, 38, 39, *rectified linear activation function*

S

SLAs 51, *Service-Level-Agreement*

Swap 20, *Auslagerungspartition falls RAM ausgelastet ist*

T

Temp 20, *Abkürzung für Temperatur*

Thermal Throttling 26, *Verringerung der Performance durch hohe Temperaturen (bei Überhitzung)*

Thread 31, *Leichtgewichtiger Prozess*

traditionellen Residualmodellen 36, *Modell, welches die Anzahl Kanäle zuerst vergrössert, dann verkleinert und am Ende wieder vergrössert*

U

USB 2.0 13, *USB 2.0 - Im Jahr 2000 spezifizierter Standard für Universal Serial Bus (480Mbps)*

14. Literaturverzeichnis

[1] A. Studer, Y. Rek und B. Peter, „JetBot - Autonomes und fortlaufendes maschinelles Lernen," OST Ostschweizer Fachhochschule, Rapperswil-Jona, 2020.

[2] tokk-nv und jaybdub, „JetBot: Hardware Setup," 22 10 2020. [Online]. Available: https://jetbot.org/v0.4.3/hardware_setup.html. [Zugriff am 21 04 2021].

[3] Ron, „thestrongestlink: Ultrasonic Sensor for the Jetbot," 31 05 2020. [Online]. Available: https://www.thestrongestlink.nl/2020/05/31/ultrasonic-sensor-for-the-jetbot/. [Zugriff am 11 03 2021].

[4] KT-Elektronic, „mikrocontroller.net," 01 2012. [Online]. Available: https://www.mikrocontroller.net/attachment/218122/HC-SR04_ultraschallmodul_beschreibung_3.pdf. [Zugriff am 01 06 2021].

[5] Nvidia, „Nvidia Developer: Jetson Nano Technical Specifications," [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano. [Zugriff am 22 04 2021].

[6] rbonghi, „GitHub: jetson_stats," 19 07 2020. [Online]. Available: https://github.com/rbonghi/jetson_stats. [Zugriff am 22 04 2021].

[7] Adafruit, „Adafruit: Triple-axis Accelerometer+Magnetometer (Compass) Board - LSM303," [Online]. Available: https://www.adafruit.com/product/1120. [Zugriff am 21 04 2021].

[8] N. Bauza, „pixabay: Oldtimer," 16 02 2016. [Online]. Available: https://pixabay.com/de/photos/oldtimer-auto-altes-auto-cabrio-1197800/. [Zugriff am 16 05 2021].

[9] „Nvidia: NVIDIA TensorRT," 2021. [Online]. Available: https://developer.nvidia.com/tensorrt. [Zugriff am 07 06 2021].

[10] G. Shekofa, S. Cemil und D. Akif, „ResearchGate: High-level diagram of SSD," 04 2019. [Online]. Available: https://www.researchgate.net/figure/High-level-diagram-of-SSD-16-for-generic-object-detection_fig2_334987612. [Zugriff am 16 05 2021].

[11] „MathWorks: Anchor Boxes for Object Detection," 2021. [Online]. Available: https://de.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html . [Zugriff am 19 05 2021].

[12] R. Girshick, J. Donahue, T. Darrell und J. Malik, „arXiv: Rich feature hierarchies for accurate object detection and semantic segmentation," 22 10 2014. [Online]. Available: https://arxiv.org/pdf/1311.2524.pdf. [Zugriff am 19 05 2021].

[13] „MathWorks: Getting Started with SSD Multibox Detection," 2021. [Online]. Available: https://de.mathworks.com/help/vision/ug/getting-started-with-ssd.html . [Zugriff am 16 05 2021].

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu und A. C. Berg, „arXiv: SSD: Single Shot MultiBox Detector," 29 12 2016. [Online]. Available: https://arxiv.org/pdf/1512.02325.pdf. [Zugriff am 18 05 2021].

[15] P. Team, „PyTorch: MOBILENET V2," 2021. [Online]. Available: https://pytorch.org/hub/pytorch_vision_mobilenet_v2/. [Zugriff am 17 05 2021].

[16] J. Brownlee, „machinelearningmastery: A Gentle Introduction to Batch Normalization for Deep Neural Networks," 16 01 2019. [Online]. Available: https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/. [Zugriff am 19 05 2021].

[17] „paperswithcode: ReLU6," 2021. [Online]. Available: https://paperswithcode.com/method/relu6#. [Zugriff am 17 05 2021].

[18] M. Hollemans, „machinethink: Google's MobileNets on the iPhone," 14 06 2017. [Online]. Available: https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/. [Zugriff am 19 05 2021].

[19] M. Lin, Q. Chen und S. Yan, „arXiv: Network In Network," 04 03 2014. [Online]. Available: https://arxiv.org/pdf/1312.4400v3.pdf. [Zugriff am 19 05 2021].

[20] J. W. Francis, „Wolfram: SSD-MobileNet V2 Trained on MS-COCO Data," 30 09 2019. [Online]. Available: https://resources.wolframcloud.com/NeuralNetRepository/resources/SSD-MobileNet-V2-Trained-on-MS-COCO-Data/. [Zugriff am 17 05 2021].

[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto und H. Adam, „arXiv: MobileNets," 17 04 2017. [Online]. Available: https://arxiv.org/pdf/1704.04861.pdf. [Zugriff am 17 05 2021].

[22] M. Sandler, A. Horward, M. Zhu, A. Zhmoginov und L.-C. Chen, „arXiv: MobileNetV2," 21 03 2019. [Online]. Available: https://arxiv.org/pdf/1801.04381v4.pdf. [Zugriff am 17 05 2021].

[23] „paperswithcode: COCO," 2021, [Online]. Available: https://paperswithcode.com/dataset/coco. [Zugriff am 16 05 2021].

[24] „cocodataset: What is COCO?," 2021. [Online]. Available: https://cocodataset.org/#home. [Zugriff am 16 05 2021].

[25] „pytorch: torchvision," 2021. [Online]. Available: https://pytorch.org/vision/stable/index.html. [Zugriff am 06 07 2021].

[26] OpenClipart-Vectors, „pixabay: Ford Mustang," 06 10 2013. [Online]. Available: https://pixabay.com/de/vectors/ford-mustang-auto-rennwagen-146580/. [Zugriff am 14 05 2021].

[27] „JetBot: Collision Avoidance," 2021. [Online]. Available: http://jetbot.org/v0.4.3/examples/collision_avoidance.html. [Zugriff am 20 05 2021].

[28] C. Yato, Jaybdub und ebuehrle, „GitHub: NVIDIA-AI-IOT: Collision Avoidance," 08 01 2021. [Online]. Available: https://github.com/NVIDIA-AI-IOT/jetbot/blob/master/notebooks/collision_avoidance/data_collection.ipynb. [Zugriff am 20 05 2021].

[29] „IMAGENET," 11 03 2021. [Online]. Available: https://www.image-net.org/. [Zugriff am 20 05 2021].

[30] „Keras: MaxPooling2D layer," 2021. [Online]. Available: https://keras.io/api/layers/pooling_layers/max_pooling2d/. [Zugriff am 20 05 2021].

[31] P. Dahal, „DeepNotes: Maxpool Layer," 2021. [Online]. Available: https://deepnotes.io/maxpool. [Zugriff am 20 05 2021].

[32] J. Brownlee, „machinelearningmastery: Softmax Activation Function with Python," 19 10 2020. [Online]. Available: https://machinelearningmastery.com/softmax-activation-function-with-python/. [Zugriff am 20 05 2021].

[33] J. Brownlee, „machinelearningmastery: A Gentle Introduction to the Rectified Linear Unit (ReLU)," 20 08 2019. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. [Zugriff am 20 05 2021].

[34] A. Anwar, „towardsdatascience: Difference between AlexNet, VGGNet, ResNet, and Inception," 07 06 2019. [Online]. Available: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96. [Zugriff am 20 05 2021].

[35] K. Patel, „towardsdatascience: Architecture comparison of AlexNet, VGGNet, ResNet, Inception, DenseNet," 08 03 2020. [Online]. Available: https://towardsdatascience.com/architecture-comparison-of-alexnet-vggnet-resnet-inception-densenet-beb8b116866d. [Zugriff am 20 05 2021].

[36] Koustubh, „CV-Tricks: ResNet, AlexNet, VGGNet, Inception,“ 03 08 2018. [Online]. Available: https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/. [Zugriff am 20 05 2021].

[37] A. Khvostikov, „ResearchGate: AlexNet architecture,“ 01 2018. [Online]. Available: https://www.researchgate.net/figure/AlexNet-architecture-Includes-5-convolutional-layers-and-3-fullyconnected-layers_fig3_322592079. [Zugriff am 20 05 2021].

[38] S. Nayak, „Learn OpenCV: Understanding AlexNet,“ 13 06 2018. [Online]. Available: https://learnopencv.com/understanding-alexnet/. [Zugriff am 20 05 2021].

[39] R. Becker, „jaai: Convolutional Neural Networks – Aufbau, Funktion und Anwendungsgebiete,“ 06 02 2019. [Online]. Available: https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/. [Zugriff am 20 05 2021].

[40] B. Ans, S. Rousset, M. R. French und S. Musca, „Taylor & Francis: Self-refreshing memory in artificial neural networks: learning temporal sequences without catastrophic forgetting,“ 21 10 2010. [Online]. Available: https://doi.org/10.1080/09540090412331271199. [Zugriff am 20 05 2021].

[41] „Microsoft Azure: App Service - Preise,“ [Online]. Available: https://azure.microsoft.com/de-de/pricing/details/app-service/windows/. [Zugriff am 07 06 2021].

15. Anhang

15.1 Roboter – Abhängigkeitsdiagramm

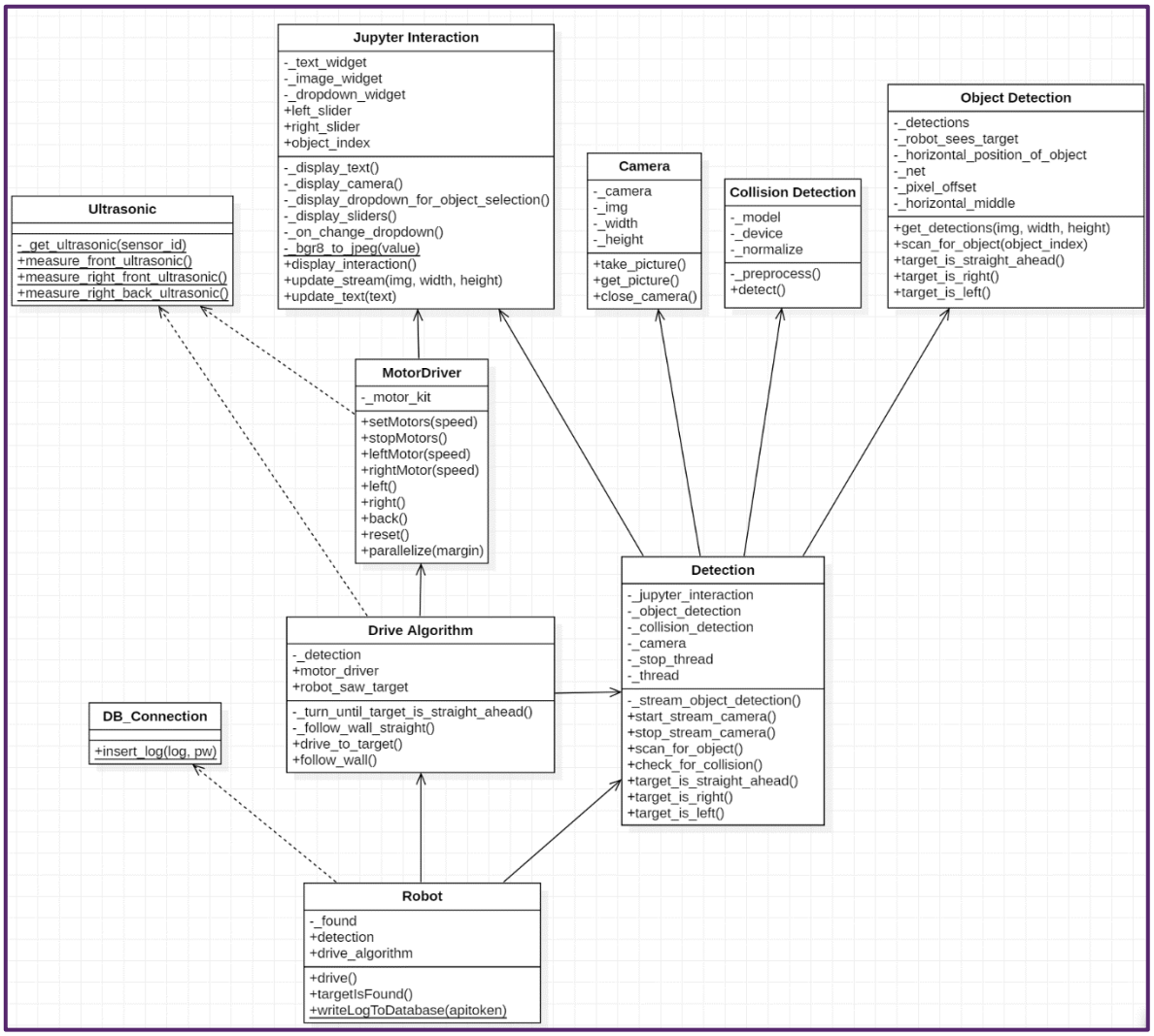


Abbildung 39: Anhang: Abhängigkeitsdiagramm JetBot [4]

[4] https://github.com/yV11/ba-fs2021/blob/main/docs/class-diagram.png

15.2 Roboter – Embedded Testing Protokoll

embedded testing

test_microcontroller_availability

test-id	steps	action	setpoint	status (pass / fail)	comment
1	1	check path /dev/ttyACM0	assert true	pass	-

test_right_back_ultrasonic_connection

test-id	steps	action	setpoint	status (pass / fail)	comment
2	1	set jetbot to test location	position in playground is set	pass	-
	2	check return value of measure_right_back_ultrasonic()	assert true (<= 700 and >= 70) (response time)	pass	-

test_right_front_ultrasonic_connection

test-id	steps	action	setpoint	status (pass / fail)	comment
3	1	set jetbot to test location	position is set	pass	-
	2	check return value of measure_right_front_ultrasonic()	assert true (<= 700 and >= 70) (response time)	pass	-

test_front_ultrasonic_connection

test-id	steps	action	setpoint	status (pass / fail)	comment
4	1	set jetbot to test location	position is set	pass	-
	2	check return value of measure_front_ultrasonic()	assert true (<= 700 and >= 70) (response time)	pass	-

test_camera_access

test-id	steps	action	setpoint	status (pass / fail)	comment
5	1	check path /dev/video0	assert true	pass	-

test_motor_controller_and_motors_and_display_connection

test-id	steps	action	setpoint	status (pass / fail)	comment
6	1	initialize robot	check that no exception occurred (assert false)	pass	-

test_wlan_connection

test-id	steps	action	setpoint	status (pass / fail)	comment
7	1	request to https://www.google.com	assert true	pass	-

test_object_detection_model

test-id	steps	action	setpoint	status (pass / fail)	comment
8	1	check if model for object detection is available	assert true	pass	-

test_collision_avoidance_model

test-id	steps	action	setpoint	status (pass / fail)	comment
9	1	check if model for collision avoidance is available	assert true	pass	-

test_collision_avoidance_model_edge

test-id	steps	action	setpoint	status (pass / fail)	comment
10	1	check if model for collision avoidance is available, for edge detection application	assert true	pass	-

test_custom_library

test-id	steps	action	setpoint	status (pass / fail)	comment
11	1	check path /AlgortihmLibrary	assert true	pass	-

date: 2021-06-07

Abbildung 40: Anhang: Embedded Testing Protokoll [5]

ba-fs2021-main-notebook.ip ● | alogrithm_log.txt | jetbot@nano-4gb-jp441: ~/j

```
1  embedded testing log - 2021-06-07 11:44:53.737336
2  test_camera_access - pass
3  test_collision_avoidance_model - pass
4  test_collision_avoidance_model_edge - pass
5  test_custom_library - pass
6  test_front_ultrasonic_connection - pass
7  test_microcontroller_availabillity - pass
8  test_motor_controller_and_motors_and_display_connection - pass
9  test_object_detection_model - pass
10 test_right_back_ultrasonic_connection - pass
11 test_right_front_ultrasonic_connection - pass
12 test_wlan_connection - pass
13
```

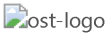
Abbildung 41: Anhang: Embedded Testing Protokoll, JupyterLab

[5] https://github.com/yV11/ba-fs2021/blob/main/docs/ba-fs-2021-testing.pdf

15.3 Roboter – Jupyter-Notebook

Bachelor Thesis FS2021

Bachelor Thesis FS2021, Computer Science @ost



Aaron Studer - aaron.studer@ost.ch

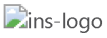
Benjamin Peter - benjamin.peter@ost.ch

Yanick Rek - yanick.rek@ost.ch

Supervisor: Prof. Dr.-Ing. Andreas Rinkel, Marc Sommerhalder

Expert: Knut Schmahl

Project partner: INS, Institute for Networked Solutions



Shortcuts

- Initialization and Testing
- Algorithmic Pathfinding + Object Detection + Collision Avoidance (Search Target))
- Collision Avoidance (Edge Detection))

1. Introduction

The main goal of this thesis is to show new students the application fields of artificial intelligence in an interactive way with the help of a robot. This is made possible with the help of a physical model and a web app. Live demonstrations can be performed in the physical test environment (this environment can also be brought to info sessions). The web app is used to explain the theory behind it.

1.1 GitHub Repo

- ba-fs2021
- ba-fs2021-web-app

1.2 Web-App

- web-app - shortcut

2. Initialization and Testing

First of all, we want to initalize the robot and run some tests to ensure the correctness of the robot.

2.1 Initialization

Abbildung 42: Anhang: Main Jupyter-Notebook, Seite 1

In order to gain access to the microcontroller, the permission needs to be set.

In [ ]:
```
import getpass
import os
import sys
sys.path.append("./AlgorithmLibrary/")
sys.path.append("./EdgeDetectionLibrary/")
```

Enter the password for the jetbot user in the input field after the execution of the following cell (the return vale should be 0):

In [ ]:
```
%%capture test
password = getpass.getpass()
command = "sudo -S chmod 777 /dev/ttyACM0"
os.system('echo %s | %s' % (password, command))
```

In [ ]:
```
import cv2
import torch
import jetbot
```

2.2 Embedded Testing

Place the robot in the playground on the marked testing section. After this step, you can run the following cells:

In [ ]:
```
from read_write_file import ReadWriteFile
ReadWriteFile.deletefile()
```

In [ ]:
```
%%capture test
!python3 embeddedTesting.py
```

Check the output from the code below, all the tests should have the label "pass" in the end of each line and they should be marked green.

In [ ]:
```
from termcolor import colored
isFailing = False
log = open("alogrithm_log.txt")
txt = log.readlines()
for line in txt:
    if(line.endswith("\n")):
        line = line[:-1]
    if(line.endswith("pass") or line.endswith("pass\n")):
        print(colored(line, 'green'))
    elif(line.endswith("fail") or line.endswith("fail\n")):
        print(colored(line, 'red'))
        isFailing=True
    else:
        print(line)
if(isFailing):
    ReadWriteFile.deletefile()
    raise SystemExit("embedded testing failed.")
log.close()
```

3. Applications

Abbildung 43: Anhang: Main Jupyter-Notebook, Seite 2

The following chapter covers all solved problems. To jump to a specific application use the index feature of JupyterLab.

3.1 Algorithmic Pathfinding + Object Detection + Collision Avoidance (Search Target)

Scenario: The robot is placed in connected rooms. While moving through the rooms, the robot searches for a previously defined object. When this target is detected, the robot moves towards it until it is directly in front of it.

Precodnitions: The robot must be placed in the marked start area and a target must be placed.

Problem Solving Strategies:

- Pathfinding: Algorithmic
- Object detection: ML
- Collision avoidance: ML

In [ ]:
```
from jupyter_interaction import JupyterInteraction

jupyter_interaction = JupyterInteraction()
jupyter_interaction.display_interaction()
```

In [ ]:
```
from robot import Robot

robot = Robot(jupyter_interaction)
```

Run the following cell to start the application.

In [ ]:
```
while(robot.targetIsFound() == False):
    robot.drive()
```

The following cell resets the robot (this step needs to be done after each execution, even if the object was not found).

In [ ]:
```
robot.drive_algorithm.motor_driver.reset()
robot.detection.stop_stream_camera()
```

The following code is used to upload the logfile from the current execution to the web app.

In [ ]:
```
apitoken = getpass.getpass()
robot.writeLogToDatabase(apitoken)
```

3.2 Collision Avoidance (Edge Detection)

Scenario: The robot is placed on a free-standing table. After placing, the robot moves freely on the surface. When a table edge is detected, a left turn is made.

Precodnitions: The robot must be placed in the middle of a free-standing table.

Problem Solving Strategies:

Abbildung 44: Anhang: Main Jupyter-Notebook, Seite 3

- Edge detection: ML (Collision Avoidance)

In [ ]:
```
from IPython.display import display
from edge_detection import EdgeDetection
import ipywidgets.widgets as widgets
```

The following cell will start the edge detection application.

In [ ]:
```
ed = EdgeDetection()
display(widgets.HBox([ed.image, ed.blocked_slider]))
ed.start()
```

To stop the application (video-stream and motors) run the following cell.

In [ ]:
```
ed.stop()
```

Side Note

In the "EdgeDetectionLibrary" folder is the notebook "collect-data and train-model" present. This is used to collect your own new data and train a new model.

Abbildung 45: Anhang: Main Jupyter-Notebook, Seite 4

15.4 Web-App – Visueller Vergleich der Versionen

Tabelle 16: Anhang: Visueller Vergleich der Web-App Versionen

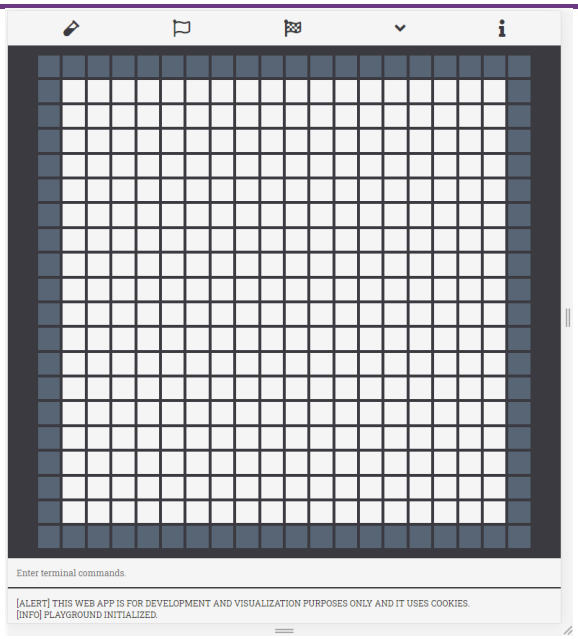
<table>
<tr><th colspan="2">/</th></tr>
<tr><td> 

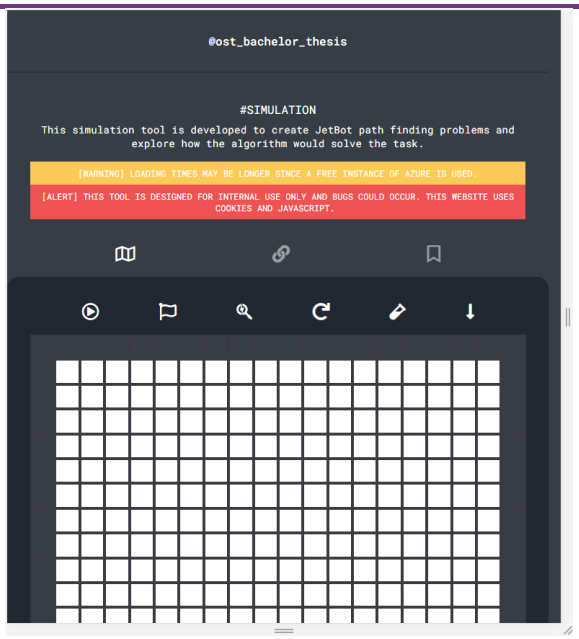
Abbildung 46: Anhang: Web-App (/) alte Version</td><td> 

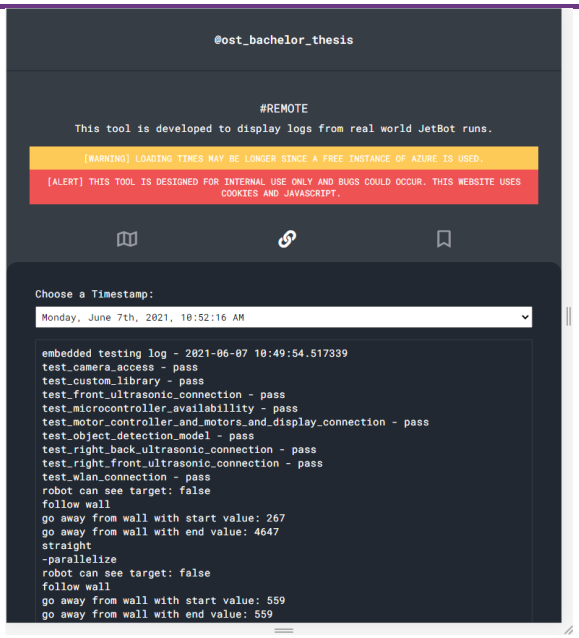
Abbildung 47: Anhang: Web-App (/) neue Version</td></tr>
<tr><th colspan="2">/remote</th></tr>
<tr><td>Kein Vergleich, dieses Feature wurde komplett neu implementiert.</td><td> 

Abbildung 48: Anhang: Web-App (/remote) neue Version</td></tr>
<tr><th colspan="2">/readme</th></tr>
</table>

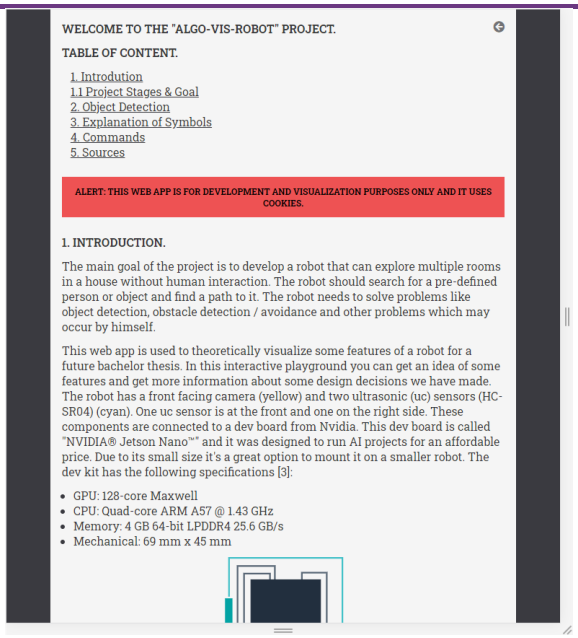


Abbildung 49: Anhang: Web-App (/readme) alte Version

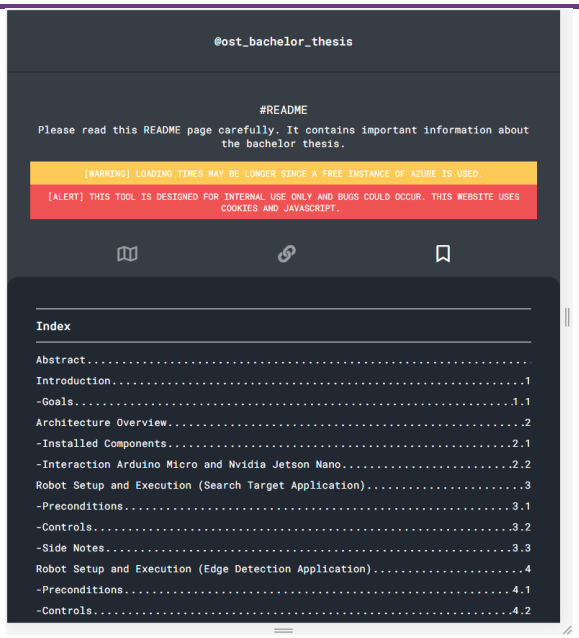


Abbildung 50: Anhang: Web-App (/readme) neue Version

- Alte Version: https://github.com/yV11/algo-vis-robot
- Neue Version: https://github.com/yV11/ba-fs2021-web-app

15.5 Web-App – Google Lighthouse Testing Protokoll

google lighthouse

desktop				
url	performance	accessibility	best practices	comment (relevant references)
/	100	65	93	**Performance:** Pass. **Accessibility:** Contrast Issues (for warning, alert and footer), links do not have a discernible name and disabling zooming is problematic. **Best practices:** Pass.
mobile				
url	performance	accessibility	best practices	comment (relevant references)
/	95	65	93	**Performance:** Pass. **Accessibility:** Contrast Issues (for warning, alert and footer), links do not have a discernible name and disabling zooming is problematic. **Best practices:** Pass.
desktop				
url	performance	accessibility	best practices	comment (relevant references)
/remote	94	71	93	**Performance:** Pass. **Accessibility:** Contrast Issues (for warning, alert and footer), links do not have a discernible name and disabling zooming is problematic. **Best practices:** Pass.
mobile				
url	performance	accessibility	best practices	comment (relevant references)
/remote	89	71	93	**Performance:** Pass. **Accessibility:** Contrast Issues (for warning, alert and footer), links do not have a discernible name and disabling zooming is problematic. **Best practices:** Pass.
desktop				
url	performance	accessibility	best practices	comment (relevant references)
/readme	99	71	93	**Performance:** Pass. **Accessibility:** Contrast Issues (for warning, alert and footer), links do not have a discernible name and disabling zooming is problematic. **Best practices:** Pass.
mobile				
url	performance	accessibility	best practices	comment (relevant references)
/readme	93	71	93	**Performance:** Pass. **Accessibility:** Contrast Issues (for warning, alert and footer), links do not have a discernible name and disabling zooming is problematic. **Best practices:** Pass.
mean	95.00	69.00	93.00	

date: 2021-06-07

bad:	0-55
sufficient - good:	56-85
very good (=pass):	86-100

Abbildung 51: Abhang: Google Lighthouse Testing Protokoll [6]

[6] https://github.com/yV11/ba-fs2021/blob/main/docs/ba-fs-2021-google-lighthouse.pdf

15.6 Web-App – Unittest, Database Testing Protokoll

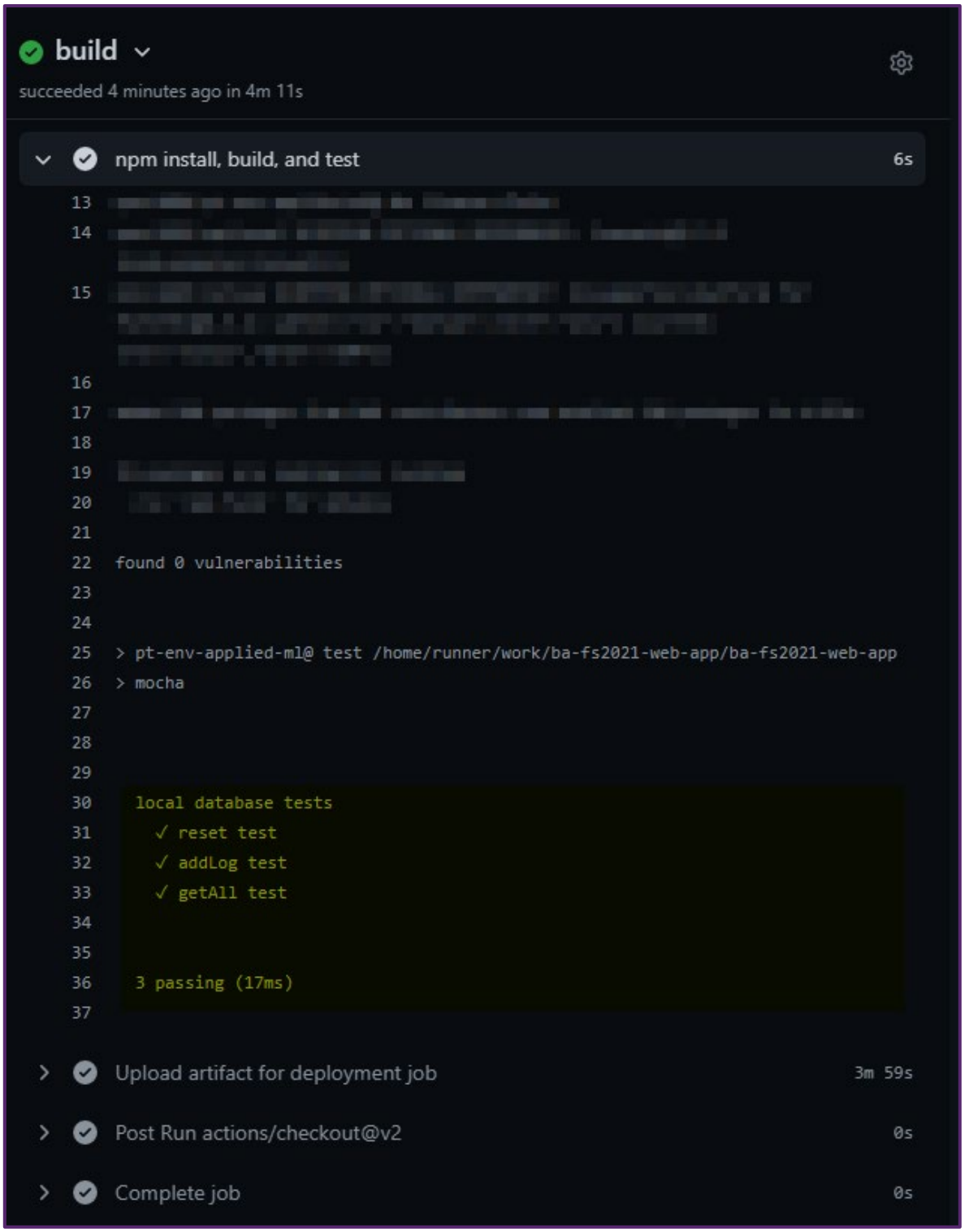


Abbildung 52: Anhang: Unittest, Testing Protokoll, 2021-06-07

15.7 Aufgabenstellung

OST
Ostschweizer
Fachhochschule

Aufgabenstellung

JetBot - Präsentationsumgebung für angewandte Machine Learning Probleme

Trotz grossen Fortschritten in weiten Bereichen der künstlichen Intelligenz sind im Bereich der angewandten künstlichen Intelligenz Themen der Autonomie und Adaption weiterhin nur wenig thematisiert. Ein allgemeiner Einsatz von Robotern für alltägliche Aufgaben ist allerdings erst möglich, wenn diese in der Lage sind, sich autonom und selbständig an Situationen anzupassen und neue Inhalte zu erlernen. Ein Pflegeroboter zum Beispiel muss sich, wenn möglich, selbständig an die Umgebung anpassen und Verhaltensweisen einzelner Patienten erlernen und interpretieren können. Es stellt sich also die Frage, wie einfache bis komplexe sensorische kognitive Leistungen auf vorhandenen Roboter-Plattformen implementiert werden können.

In dieser Arbeit soll ein erster Schritt in Richtung Autonomie auf der Grundlage des JetBots unternommen werden. Der JetBot ist ein einfacher und kostengünstiger Roboter mit Kamera auf zwei Rädern, welcher über Python/Jupyter/Webserver programmiert werden kann. Da der JetBot auf dem Jetson Nano von NVIDIA aufbaut, eignet er sich insbesondere für die Arbeit mit neuronalen Netzen. Von Haus aus kann der JetBot bereits einfach manövriert werden. Kamerabilder können ausgelesen und unter der Verwendung bereits existierender Modelle bezüglich Kollision und Objekterkennung analysiert werden. Im Rahmen dieser Arbeit sollen Applikationen entwickelt werden, welche zur Veranschaulichung von Machine Learning Problemen genutzt werden können.

Rapperswil 18.06.2021,

Prof. Dr.-Ing. Andreas Rinkel	
Marc Sommerhalder	

Namen Unterschriften

Aufgabenstellung Seite 1

Abbildung 53: Anhang: Aufgabenstellung

15.8 Eigenständigkeitserklärung

OST
Ostschweizer
Fachhochschule

Eigenständigkeitserklärung

Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil 18.06.2021,

Aaron Studer	
Benjamin Peter	
Yanick Rek	

Namen Unterschriften

Eigenständigkeitserklärung Seite 1

Abbildung 54: Anhang: Eigenständigkeitserklärung

15.9 Einverständniserklärung

OST
Ostschweizer
Fachhochschule

Einverständniserklärung Publikation auf eprints.hsr.ch

O SA
☑ BA

Titel der Arbeit:	JetBot - Präsentationsumgebung für angewandte Machine Learning Probleme
Team:	Aaron Studer, Benjamin Peter, Yanick Rek
Betreuer:	Prof. Dr.-Ing. Andreas Rinkel, Marc Sommerhalder

Wir sind mit der Publikation unserer Arbeit auf eprints.hsr.ch einverstanden, sofern für diese Arbeit keine Geheimhaltungsvereinbarung unterzeichnet wurde.
Nach Bekanntgabe der Note haben wir die Möglichkeit innert 14 Tagen Einsprache zu erheben und das Einverständnis zur Publikation der Arbeit auf eprints.hsr.ch zurückzuziehen. In diesem Falle wird nur der Abstract publiziert.

Rapperswil 18.06.2021,

Aaron Studer	
Benjamin Peter	
Yanick Rek	

Namen Unterschriften

Einverständniserklärung Publikation auf eprints.hsr.ch Seite 1

Abbildung 55: Anhang: Einverständniserklärung

15.10 Vereinbarung über Urheber- und Nutzungsrechte

OST
Ostschweizer
Fachhochschule

Vereinbarung über Urheber- und Nutzungsrechte

1. Vereinbarung

1.1 Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Bachelorarbeit «JetBot - Präsentationsumgebung für angewandte Machine Learning Probleme» von Aaron Studer, Benjamin Peter und Yanick Rek unter der Betreuung von Prof. Dr.-Ing. Andreas Rinkel und Marc Sommerhalder geregelt.

1.2 Urheberrecht

Die Urheberrechte stehen den Studenten zu.

1.3 Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von den Studenten, von der OST wie vom INS, Institute for Networked Solutions nach Abschluss der Arbeit verwendet und weiterentwickelt werden.

Rapperswil 18.06.2021,

Aaron Studer	
Benjamin Peter	
Yanick Rek	

Prof. Dr.-Ing. Andreas Rinkel	
Marc Sommerhalder	

Namen Unterschriften

Abbildung 56: Anhang: Vereinbarung über Urheber- und Nutzungsrechte, Seite 1

OST Ostschweizer Fachhochschule

2. Vereinbarung

Ohne anderslautende Vereinbarungen stehen die Schutzrechte und das Know-how an der Bachelorarbeit (nachfolgend ‚Arbeit' genannt) und an der in diesem Rahmen geschaffenen Güter, wie Software, sowohl dem Rechtsträger der OST Ostschweizer Fachhochschule, dem für die Arbeit verantwortlichen Professoren sowie dem Verfasser der Arbeit resp. Entwickler der in diesem Rahmen geschaffenen Güter, wie Software, zu.

Die genannten Parteien übertragen sich gegenseitig nicht exklusiv, jedoch unentgeltlich, weltweit, sachlich und zeitlich unbeschränkt die jeweiligen Schutzrechte und das Know-how an der Arbeit und an der in diesem Rahmen geschaffenen Güter, wie Software, einschliesslich dem Recht zur Weiterübertragung, ab. Entsprechend steht es jeder Partei zu, sämtliche Schutzrechte an der Arbeit resp. an der in diesem Rahmen geschaffenen Güter, wie Software, beliebig weltweit, zeitlich und sachlich unbeschränkt zu verwerten. Darunter fällt namentlich aber nicht abschliessend das Recht zur Lizenzierung in jeder Art, Umfang und Form, das Recht zur Bearbeitung und damit zur Nutzung z. B. der Software oder Komponenten hiervon als Grundlage eines neuen schutzfähigen Guts. Die Parteien erklären sich gegenseitig den Verzicht auf Namensnennung bei der Verwertung der Schutzrechte und des Know-how durch eine oder mehrere Parteien gemeinsam und stimmen namentlich zu, dass jede Partei allein unter ihrem eigenen Namen die Schutzrechte resp. das Know-how verwertet. Die vorliegende gegenseitige unentgeltliche Übertragung der Schutzrechte resp. des Know-how bezieht sich auch auf Verwertungsarten, welche heute noch nicht bekannt sind.

Rapperswil 18.06.2021,

Aaron Studer	
Benjamin Peter	
Yanick Rek	

Prof. Dr.-Ing. Andreas Rinkel	
Marc Sommerhalder	

Namen Unterschriften

Abbildung 57: Anhang: Vereinbarung über Urheber- und Nutzungsrechte, Seite 2

ins
INSTITUTE FOR
NETWORKED SOLUTIONS

OST
Ostschweizer
Fachhochschule

Impressum

18.06.2021

Aaron Studer, Benjamin Peter, Yanick Rek

OST – Ostschweizer Fachhochschule
Studiengang Informatik

Oberseestrasse 10, Postfach
8640 Rapperswil, Switzerland

aaron.studer@ost.ch
benjamin.peter@ost.ch
yanick.rek@osr.ch
ost.ch