

# rosBerry - ein autonomer Roboter auf Basis von ROS und eines Modellautos mit Maker-Elektronik

1<sup>st</sup> Welter, Heike  
Heike.Welter@stud.hs-mannheim.de

2<sup>nd</sup> Matheis, Steffen  
Steffen.Matheis@stud.hs-mannheim.de

3<sup>rd</sup> Barsalou, Marie  
Marie.Barsalou@stud.hs-mannheim.de

**Zusammenfassung**—Ziel von Projekt rosBerry besteht darin, einen kleinen, schnellen aber dennoch preisgünstigen Roboter aus leicht verfügbaren Standardbauteilen zu bauen. Zusätzlich soll er mit dem Robot Operating System betrieben und mit einer Künstlichen Intelligenz versehen werden. Der Roboter soll einen optischen Marker erkennen und darauf zufahren. Frontalzusammenstöße mit Wänden soll er dabei vermeiden.

**Index Terms**—Roboter, ROS, Raspi, Neuronale Netze, Arduino

## I. THEORIE ZU ROBOTER UND SEINER KÜNSTLICHEN INTELLIGENZ

Wenige Projekte in Robotik oder Künstlichen Intelligenz für Autonome Systeme existieren, ohne das es nicht andere vergleichbare Projekte gibt. Eine vollständige oder zumindest Repräsentative Darstellung würde den Rahmen einer Studentischen Projektarbeit sprengen.

### A. Vergleichbare Hardwareansätze — Donkey Cars

Die Hardware des Roboters wurde vom Donkey Car Projekt inspiriert. Das Donkey Car Projekt beschäftigt sich damit, ein ferngesteuertes Auto mit einem Raspberry Pi zu steuern. Mehr Informationen zu diesem Projekt gibt es auf ihrer Website <https://docs.donkeycar.com>. Donkey Cars fahren größtenteils mit Hilfe von quelloffenem Python Code.

Auf Youtube hat ein Benutzer namens Tiziano Fiorenzani gezeigt, wie ein Donkey Car mit ROS betrieben werden kann. Das Video [https://youtu.be/iLiI\\_IRedhI](https://youtu.be/iLiI_IRedhI) verweist auf Github für Tiziano Fiorenzanis Code. Es wurde eine Abzweigung eines Github-Repositorys erstellt. Auf [https://github.com/Wifi-cable/Robotic\\_AI\\_student\\_project/](https://github.com/Wifi-cable/Robotic_AI_student_project/) Findet sich der Quellcode des Projektes. Das Projekt wurde um einen Arduino und mehrere ROS Knoten sowie um Launchfiles erweitert.

### B. Vergleichbare Projekte zur Künstlichen Intelligenz

Das Neuronale Netz zur Bildklassifizierung ist an das Buch „Artificial Intelligence for Robotics“ [2] angelehnt. Darin vermittelt F. Grovers, wie Künstliche Intelligenz für Autonome Systeme funktionieren kann, indem er zeigt, wie er einem kleinen Roboter beibringen würde aufzuräumen. In einem Kapitel beschreibt der Autor, dass der Roboter mit Hilfe von Neuronalen Netzen lernen soll, ob sich Spielzeuge auf dem Teppich befinden, um sie später aufzuräumen.



Abbildung 1. Der Optische Marker den die KI suchen wird

Auch die Bachelorarbeit „Bildklassifikation auf einem Raspberry Pi Zero am Beispiel einer Ladestationserkennung“ [1] von Amanda Decker ist ein vergleichbares Projekt. Einige der Grundansätze aus dieser Arbeit wurden für dieses Projekt übernommen. So haben wir das Neuronale Netz nicht auf einem Raspberry Pi trainiert und einen Marker genommen der sich gut rotieren und spiegeln lässt. Somit kann jedes Bild des Datensatzes gespiegelt werden, um mit mehr Daten arbeiten zu können.

## II. THEORETISCHE GRUNDLAGEN VON PROJEKT ROSBERRY

In diesem Kapitel wird auf die Erstellung des Datensatzes für die KI eingegangen sowie auf die Grobübersicht des Softwaresystems auf die Knotenarchitektur.

### A. Ansätze für die Erstellung des Datensatzes

Bei Amanda Deckers Bachelorarbeit schien die KI Schwierigkeiten damit zu haben, zwischen dem Marker

und einem Blauton zu unterscheiden, der auch im Marker vorkam. Es schien auch schwierig zu sein ein Binärbild vom Marker zu erzeugen, da die beiden Farben des Markers einen geringen Kontrast zueinander haben. Da Kameras in Opencv mit Hilfe von Schachbrettmustern kalibriert werden, wird für dieses Projekt ein Symbol mit ähnlich hohem Kontrast und scharfen Kanten verwendet.

Dieses Symbol<sup>1</sup> ist nicht nur in 4 Richtungen rotierbar, sondern auch zweifach spiegelbar, ohne den Marker zu verfälschen. (Rotationsinvarianz und Spiegelungsinvarianz) Es verfügt auch über einen maximalen Kontrast und klare, gerade Kanten.

Ein Neuronales Netz kann nur so gut sein wie die Daten, mit denen es trainiert und getestet wurde. In vielen wissenschaftlichen und populärwissenschaftlichen Artikeln wurde beschrieben, wie ein ungeeigneter Datensatz zu unbrauchbaren Ergebnissen führte. So wurden Vorurteile der Forscher bestätigt, oder die KI traf Entscheidungen anhand von falschen Kriterien, wie der Bildunterschrift statt des Bildinhalts.

Um einen möglichst robusten Datensatz zu bekommen, werden verschiedene Techniken angewendet:

- Um zu verhindern, dass die KI alle Bilder mit starken Kontrasten für einen Marker hält, wird ein Ausschnitt einmal mit und einmal ohne den Marker fotografiert.
- Gegen den „schön-wetter KI“-Effekt (eine KI die annimmt jedes gut ausgeleuchtete Bild muss ein Treffer sein) wird der Marker bei verschiedenen Beleuchtungen fotografiert. Dazu gehören direktes Sonnenlicht, Schatten, künstliche Beleuchtung mit LED, Lampen und Neonröhren.
- Um zu verhindern, dass die KI alles für ihren Marker hält, was eine gewisse Größe hat und quadratisch ist, wurde der Marker aus verschiedenen Distanzen fotografiert.
- Gegen Noise-Anfälligkeit wurden verschiedene Hintergründe bei den Fotos verwendet. Teils weiße Wände, teils strukturierte Hintergründe oder eine Freifläche.
- Um zu verhindern, dass sich die KI auf die Bildmitte konzentrieren kann, wird der Marker aus verschiedenen Positionen fotografiert.

### B. Grobübersicht über das System

In diesem Kapitel werden wir, bevor wir näher auf die Softwarearchitektur und den Hardwareaufbau von rosBerry eingehen, das System kurz als Gesamtüberblick vorstellen.

Die Abbildung 2 zeigt das System, welches auf mehreren Komponenten basiert. Innerhalb des Systems wird eine Software eingesetzt, die auf dem Robot Operating System (ROS) basiert. Diese Software läuft verteilt auf den Komponenten Laptop, Raspberry PI und Arduino, wie in Abbildung 2 zu sehen.

Beginnen werden wir mit der Komponente Laptop. Sie ist dazu da, dem User die Steuerung des Roboters zu ermöglichen. Die Steuerung kann erst erfolgen, wenn auf dem Raspberry Pi ein Accesspoint geöffnet wurde, mit dem sich der Laptop — der Nutzer letztendlich — via WLAN verbindet.

img/Gesamtsystem.PNG

Abbildung 2. Gesamtüberblick des Systems

Innerhalb der anderen Komponente Roboter — unser rosBerry — befindet sich die beiden Hardware-Elemente Raspberry Pi und ein Arduino. Roscore läuft auf dem Raspberry PI und ein Node - für den Ultraschallsensor - auf dem Arduino.

Der Arduino wird durch ein USB Kabel mit dem Raspberry Pi verbunden. Dadurch wird eine Kommunikation ermöglicht, in welcher der Arduino den Raspberry Pi in Zeitabständen mit Sensordaten versorgt.

An dieser Stelle ist anzumerken, dass weitere Hardware verwendet wird, die im Kapitel Hardwareaufbau näher vorgestellt wird.

### C. Architektur der ROS-Knoten

In diesem Kapitel wird die Software-Architektur des Roboters vorgestellt.

Treu nach den Prinzipien von ROS besteht die Software aus vielen ROS Nodes, welche als Oval dargestellt werden. Jeder Node erledigt kleine Aufgaben und kann als Subscriber und/oder Publisher fungieren. Die Kommunikation zwischen Subscriber und Publisher erfolgt durch ROS Messages (Nachrichten). Diese werden durch verschiedene Topics — werden als Rechteck dargestellt — kommuniziert. Die Topics kann man als Kanäle für Nachrichten betrachten.

Folgende Nodes befinden sich im Softwaresystem:

- /rosout (Master)
- /teleop\_twist\_keyboard (Fernsteuerung)
- /donkey\_llc (Hardwarenahe Steuerung)
- /ic2pwm\_board\_node (Bibliothek und Node zum Ansprechen des I2C-Protokolls und deren Verbindung mit ROS)
- serial\_node.py (Publisher der Ultraschallsensordaten)

img/rosgraph.png

Abbildung 3. Übersicht der ROS Elemente

Die zuvor genannten Nodes sind durch Topics miteinander verbunden und tauschen untereinander Nachrichten aus. Folgende Topics befinden sich im Softwaresystem:

- /cmd\_vel (command velocity, die Beschleunigungssteuerung)
- /servos\_absolute (Steuerung des Servos mit absoluten Impulsstart- und Stoppwerten)
- /servos\_proportional (Motorsteuerung für Geschwindigkeitsteuerung des Servos in seinem Bewegungsbereich)
- /servos\_drive (Lenken, Umwandlung der Berechnungen von Linear- und Winkeldaten in Servoantriebsdaten)
- /rosout (Standard ROS Output)
- /rangeMsg\_topic (Datenübertragung der Ultraschallsensordaten)

1) *Launchreihenfolge:* Dieses Kapitel ist eine Einleitung zum Starten der ROS Nodes und den hierbei zu beachtenden Punkten. Ohne Vorkonfiguration und Launchfiles ist die Knotenaufreihenvolge langwierig. Manche Schritte müssen auf einem Laptop durchgeführt werden, andere via SSH auf dem Raspberry Pi. So muss Laptop erfahren das der ROScore (Master Node) bereits auf dem Raspberry Pi läuft.

---

```
# Network ubiquityrobot03B
# connect via SSH
ssh ubuntu@10.42.0.1
# enter password

# 1. Console - on Raspberry Pi
# Login via SSH
cd catkin_ws/Robotic_AI_student_project/
# If bash.rc wasn't changed to source
→ workspace
source devel/setup.bash
roslaunch i2cpwm_board i2cpwm_board

# 2. Console - on Raspberry Pi
# Login via SSH
cd catkin_ws/Robotic_AI_student_project/
source devel/setup.bash
roslaunch donkey_car low_level_control.py

# 3. Console - on Laptop
export ROS_MASTER_URI=http://ubiquityrobot03B:11311
→ t.local:11311
export ROS_IP=$(hostname -I)
cd catkin_ws/Robotic_AI_student_project/
source devel/setup.bash
roslaunch teleop_twist_keyboard
→ teleop_twist_keyboard.py

# 4. Console - on Raspberry Pi
# Login via SSH
roslaunch roserial_python serial_node.py
→ /dev/ttyACM0

# 5. Console - on Laptop
export ROS_MASTER_URI=http://ubiquityrobot03B:11311
→ t.local:11311
export ROS_IP=$(hostname -I)
rostopic echo rangeMsg_topic
```

---

Mehrere Schritte sind Möglich um diesen Prozess zu vereinfachen. Die meisten Optionen sind eine frage der persönlichen Präferenz des ROS Benutzers. So kann ein Eintrag in die Systemweit geltende Datei *bash.rc* die Eingabe von *source devel/setup.bash* im ROS workspace ersetzen.

Eine weitere Erleichterung ist der Eintrag eines Shellscripts in die *setup.bash* Datei. 4 Dieser Eintrag Setzt die ROS Host IP Variable auf die aktuelle IP Adresse des Laptops.

Eine weitere Möglichkeit der Vereinfachung besteht im erstel-

```
export ROS_IP=$(ip -br -4 address show dev wlp3s0 |
↪ awk '{print $3}' |
awk -F '/' '{print $1}')
```

Abbildung 4. Bash Einzeiler, Autor J.K

```
#call on Raspbeery Pi to launch hardware nodes
roslaunch donkey_car keyboard_demo.launch
#call keyboard node from laptop
roslaunch teleop_twist_keyboard
↪ teleop_twist_keyboard.py
```

Abbildung 5. Tastatur Steuerung per Launchfile

len von SSH-Keys.

2) *Launchfiles*: Launchfiles die viele ROS Knoten auf einmal starten sind eine Beliebte Vereinfachung. Um die *Keyboard Demo 5* zu starten, oder den Roboter Fernzusteuern existiert ein Launchfile im *Donkey Car* Fork des Github Quelle.

Um die benötigten Knoten Für die Künstliche Intelligenz komfortabel zu starten wurden zwei Launchfiles geschrieben 6 . Das ist Notwendig da die Anwendung verteilt läuft. (siehe spätere Artikel)

### III. PRAKTISCHE PROJEKTDURCHFÜHRUNG

In diesem Kapitel wird der Hardwareaufbau des *rosBerry*, sowie die mechanisches Bauteile inklusive Entscheidungskriterien vom Roboter beschrieben. Hierfür werden die einzelnen verwendeten Hardwarekomponenten — deren Modell und Funktion — vorgestellt.

Zuerst werden die Verbindungen der Hardwarekomponenten untereinander abgebildet, wie in Abbildung 9 zu sehen.

#### A. Mechanische Bauteile

Für das Robotik Projekt *rosBerry* wurde ein solides, aber günstiges Chassis gesucht bei dem die Möglichkeit besteht Ersatzteile zu bestellen. Viele qualitativ hochwertige Modellbau Chassis liegen in einem Preisrahmen über 400€, für diese ist die Ersatzteilbeschaffung gut möglich. Günstige, meist asiatische Modellbau Chassis Hersteller liefern ihre Modelle und Bausätze nur für einen kurzen Zeitraum. Es ist oft nicht möglich zu einem späteren Zeitpunkt das Modell nachzukaufen oder Ersatzteile zu bekommen. Somit ist ein Roboter der

```
#call from Raspberry Pi
roslaunch donkey_car hardware.launch
#call from Laptop
export
↪ ROS_MASTER_URI=http://ubiquityrobot.local:11311
roslaunch ki_nav start_AI.launch
```

Abbildung 6. Künstliche Intelligenz starten per Launchfile

Abbildung 7. Auszug aus Hersteller Datenblatt:Lenkservo

Abbildung 8. Auszug aus Hersteller Datenblatt: Differenzialgetriebe

img/Hardwarekomponenten.PNG

Abbildung 9. Verbindungen der Hardwarekomponenten im *rosBerry*

darauf aufbaut, wie das *Donkey Car*, nicht reproduzierbar.

Der Kompromiss dieser Abwägung fand sich bei einem älteren Einsteigermodell namens *TAMIYA RC 1:10 TT-01E*. Tamiya ist ein traditionsreicher japanischer Hersteller der qualitativ hochwertige Bausätze ferngesteuerte Fahrzeuge herstellt. (Er stellt Schiffe, Flugzeuge und Panzer her, aber hauptsächlich Modellautos) Tamiya ist unter Modellbau-Enthusiasten dafür bekannt, alle Prinzipien japanischer Ingenieurskunst bei der Konstruktion seiner Modelle anzuwenden, statt *Spielzeugqualität* in China herzustellen. Das TT01-Modell hat identische Außenmaße und kann mit denselben elektronischen Komponenten betrieben werden, wie sein Nachfolgemodell das TT02. Somit ist die Reproduzierbarkeit im Bezug auf das Chassis gegeben.

Die Mechanik des *TAMIYA RC 1:10 TT-01E* Modells basiert auf einem Motor und einer langen Antriebswelle, die drei Differenzialgetriebe bewegt. 8 Die Modelle werden als Vierradantrieb vermarktet. Die Lenkung wird Getrennt vom Antrieb durch einen Kleinen Servo Motor geregelt7. Er bewegt die Spurstange nach rechts/links. Das Chassis nutzt also eine Akkerman Lenkung, wie sie auch bei PKWs üblich ist.

#### B. Elektronische Bauteile und ihre Funktion

Wie aus der Abbildung 9 zu erkennen, ist die meiste Hardware mit Jumper Wire verbunden. Ausnahmen der Anschlüsse bilden die Verbindungen des Raspberry Pis zur Kamera (Flexkabel), zum Arduino (USB) und zum

Motorshield (I2C).

Als Nächstes wird rosBerry mit den Hardwaremodellen vorgestellt.

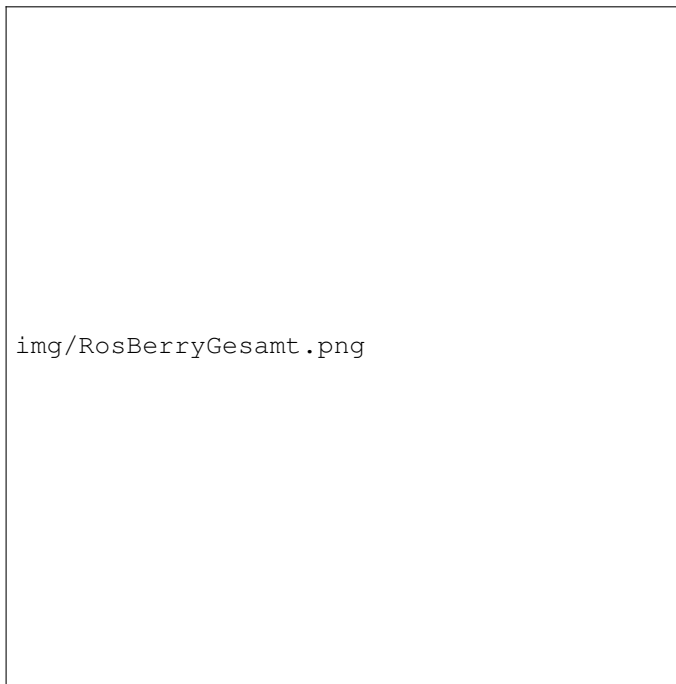


Abbildung 10. Gerüst rosBerry

Die verwendete Hardware ist in Abbildung 10 mit Nummern versehen. Auf jede dieser Nummern wird referenziert und die Hardware mit Modell und Funktion aufgelistet.

- 1) Der DC-Antriebsmotor — Modell Tamiya Mabuchi RS 540 SH — steuert alle vier Räder des rosBerry an. Ohne ihn wird der Roboter nicht in Bewegung kommen.
- 2) Neben dem DC Antriebsmotor befindet sich ein Fahrtenregler, Modell BORSTI 1/10 BRUSHED-ESC 45A. Dieser erhält vom Motorshield ein PWM-Signal, welches er verstärkt. Das verstärkte PWM-Signal wird anschließend in Geschwindigkeit umgesetzt und an den DC-Antriebsmotor weitergeleitet.
- 3) Der UBEC — Modell HWBEC Hobbywing 3A 5 V 6 V max 5 A — wird eingebaut, um den Raspberry Pi mit 5 V zu versorgen. Ohne den UBEC würde der Raspberry Pi aufgrund der höheren elektrischen Spannung beschädigt werden.
- 4) Der Servo — Modell Amewi AMX Racing 4806HB — bekommt vom Motorshield ein PWM Signal übermittelt. Nach dem PWM Signal richten sich die beiden Reifen aus und ermöglichen eine Fahrtrichtung. Dadurch ist rosBerry in der Lage links/rechts Kurven sowie geradeaus zu fahren.
- 5) Der Akku dient dazu, die Stromversorgung des Roboters zu sichern. Dieser hat eine Nennspannung von 7.2 Volt.

img/RosBerryWeitere.png

Abbildung 11. Weitere Hardwareelemente

Er versorgt die Hardware mit Strom.

- 6) Der Motorshield — Modell Adafruit 16-Channel 12-bit PWM/Servo Driver-I2C interface PCA9685 — wird für die Geschwindigkeit und die Richtung der Reifen eingesetzt. Hierzu erzeugt der Motorshield zwei PWM Signale. Das 1. PWM Signal wird an den Servo übermittelt, um den Servo in die gewünschte Position einzustellen. Das 2. PWM Signal wird erzeugt und zum Fahrtenregler gesendet. Dieses wird zur Einstellung der Geschwindigkeit verwendet.
- 7) Der Raspberry Pi 3.B+ ist das Herzstück des rosBerry. Auf ihn läuft zum einem ROS und zum anderen die Software des Roboters, welche im Kapitel ROS Knoten Software Architektur vorgestellt wurde. Über eine I2C-Schnittstelle werden Daten an den Motorshield übertragen. Die Daten geben an, wie das PWM Signal auszusehen hat.
- 8) Der Arduino UNO REV 3 steuert einen Ultraschallsensor an und erzeugt einen ROS-Knoten (Publisher). Innerhalb des Programms wird das Signal des Sensors in Meter umgewandelt und in eine Range Message gespeichert. Die Range Message namens rangeMsg wird an das Topic rangeMsg\_topic übermittelt. Ein Subscriber — der auf dem Raspberry Pi läuft — subscribed das Topic und erhält regelmäßig Sensordaten über die Reichweite.
- 9) Der Ultraschallsensor — Modell HC-SR04 — wird zum rechtzeitigen Erkennen eines kommenden Hindernisses eingesetzt. Dieser hat eine maximale Reichweite von vier Meter, jedoch haben wir uns für das Projekt auf eine Reichweite von 2 Meter geeinigt. Alle 1/4 Sekunde sendet dieser ein Signal und verarbeitet dieses.
- 10) Die Kamera — Modell 5 MP — ist durch ein Flexkabel an den Raspberry Pi über das CSI-Interface angeschlossen. Diese versorgt die KI mit Bildern, welche einen Marker sucht und findet.



#### IV. AUTONOMES VERHALTEN DURCH KÜNSTLICHE INTELLIGENZ

Ziel des KI-Teils des Projektes besteht darin, dem Roboter beizubringen, einem optischen Marker zu folgen. Dazu muss er erst einmal das Symbol erkennen, dann herausfinden in welcher Richtung sich der Marker befindet. Danach erst kann er auf den Marker zufahren.

Der schwierigste Teil der Aufgabe besteht darin, den Marker zuverlässig in unterschiedlichen Umgebungen zu erkennen.

##### A. Erster Ansatz für das Neuronale Netz

Der erste Entwurf unseres Neuronalen Netzes war stark angelehnt an F. Grovers Beispielnetz aus „Artificial Intelligence for Robotics“[2]. Wie in der Vorlage erstellen wir ein sequentielles Modell mithilfe von Keras.

Die erste Schicht stellt ein convolution layer mit 20 convolutions dar, wovon jede ein Merkmal des Eingangsbildes isolieren soll. Die Größe setzen wir auf 5x5, betrachten also zwei benachbarte Pixel in jede Richtung. Selbstverständlich muss dafür auch ein Padding hinzugefügt werden, damit das Verfahren auch am Bildrand funktioniert.

Zudem wird noch eine Aktivierungsfunktion benötigt, hierfür nutzen wir die ReLU-Funktion, die nur positive Werte zulässt und negative Werte durch den Wert 0 ersetzt. Diese Aktivierungsfunktion ist deshalb sinnvoll, da das Ergebnis als Farbe dargestellt werden und deshalb einen Wert zwischen 0 und 1 annehmen soll. Ein Wert unter Null wäre hier unbrauchbar.

```
model = Sequential()
inputShape = (height, width, depth)

model.add(Conv2D(20, (5, 5),
    → padding="same",
    → input_shape=inputShape))
model.add(Activation("relu"))
```

Die zweite Schicht ist ein Maxpooling-Layer, bei dem wir jeweils 2x2 Pixel nehmen und danach 2x2 Pixel weiterrücken um keine Überlappungen zu haben. Das Ergebnis ist ein Bild in ¼ der originalen Größe: aus 640x480px wird 320x240px, aus 128x128px 64x64px usw.

Direkt darauf folgt ein weiteres convolution layer mit der doppelten Anzahl an convolutions — 40 statt vorher 20 — um mehr Merkmale zu identifizieren. Durch das vorher durchgeführte maxpooling werden nun andere, größere Merkmale erkannt. Ebenfalls nutzen wir wieder dieselbe ReLU Aktivierungsfunktion, aus denselben Gründen wie in der ersten Schicht.

```
model.add(MaxPooling2D(pool_size=(2, 2),
    → strides=(2, 2)))
model.add(Conv2D(40, (5, 5),
    → padding="same"))
model.add(Activation("relu"))
```

Schicht vier besteht erneut aus einem Maxpooling-Layer, ebenfalls um nochmals größere Merkmale zu erkennen. Aus einem 640x480px großen Bild wird nun 160x120px, aus 128x128px sogar nur noch 32x32px.

Darauf folgt nun eine Schicht, die vorher noch nicht im Neuronalen Netz vorkam: Zuerst werden die dreidimensionalen Bildinformationen mit der Funktion Flatten() in ein eindimensionales Array geplättet, bevor wir eine vollverknüpfte Schicht mittels Dense(500) hinzufügen. Erneut nutzen wir die ReLU Aktivierungsfunktion, aus bekannten Gründen.

Listing 1. Modell: vierte und fünfte Schicht

```
model.add(MaxPooling2D(pool_size=(2, 2),
    strides=(2, 2)))

model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))
```

Abschließend benötigen wir zwei Neuronen für die beiden Ausgangswerte "Marker" und "kein Marker". Zur Aktivierung nutzen wir die Softmax-Funktion, die auch normalisierte Exponentialfunktion genannt wird. Sie transformiert mehrdimensionale Vektoren in einen Wertebereich zwischen 0 und 1, wobei alle Komponenten zusammen 1 ergeben müssen, das heißt wenn "Marker" einen Wert von 0,65 hat, muss "kein Marker" einen Wert von 0,35 haben. Die Funktion kann durchaus für mehr als zwei Ausgangswerte genutzt werden, für unseren Anwendungsfall werden jedoch nicht mehr Ausgangswerte benötigt.

Listing 2. Modell: finale sechste Schicht

```
model.add(Dense(2))
model.add(Activation("softmax"))
```

##### 1) Erste Auswertung der Ergebnisse: //kommt noch

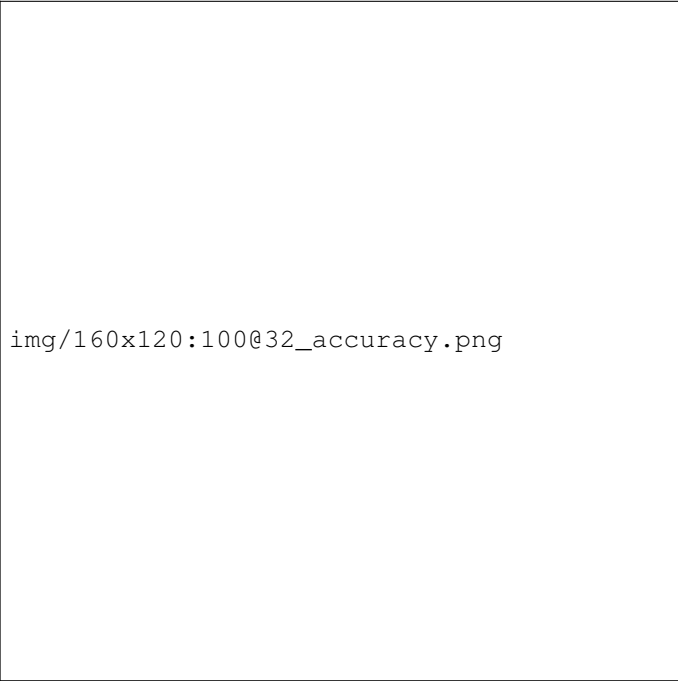
##### B. Schritte zur Verbesserung der Erkennungsrate

Beim ersten Durchlauf des Neuronalen Netzes war die Erkennungsrate des Markers 57%. Dieses Ergebnis ist bei einer 50/50 Chance kaum besser als Raten.

Die Schritte von „Learn Keras for Deep Neural Networks“[4] zum verbessern des Designs von Neuronalen Netzen wurden zur verbesserung der Trainignsergebnisse verwedet.

[4] Rät dazu mit einer kleinen Architektur anfangen das bedeutet wenige Schichten und wenige Neuronen. Es liegt nahe das kleine Architekturen Ressourcen schonender sind als grosse Architekturen. Dieser ansatz erscheint sinnvoll da Steigende Ressourcenanforderngen bei gleichbleibender Hardware zu steigender Rechenzeit pro Epoche bedeuten.


Das Netz nutzte anfangs nur eine Auflösung von 128x128 Pixel und 100 Epochen. Da jedes einzelne Pixel eine Eingangsneurone wird, ist anzunehmen, dass die geringe



img/160x120:100@32\_accuracy.png

Abbildung 12. erster Durchlauf

Auflösung im Buch gewählt wurde, um Rechenleistung zu sparen.



img/213x160:400@32\_accuracy.png

Abbildung 13. overfitting

[4] Rät die Anzahl an Neuronen pro Schicht erhöhen, wenn

Anfangs kein befriedigenden Ergebnisse erzielt werden. Bei voller Bildauflösung von 640x480 Pixel erweist sich das Neuronale Netz auf der Hardware eines handelsüblichen Laptops als nicht mehr lauffähig. Für weitere Durchläufe wurden die nächsten Versuche auf einer Workstation mit mehr Grafikkarten und Deutlich mehr RAM gemacht. Die Erkennungsrate stieg bei größerer Auflösung und mehr Durchläufen tatsächlich auf über 80%. Damit vergrößerte sich jedoch das Modell (die Gewichte der KI) auf über ein Gigabyte. Ein so großes Modell kann nicht zeitnahe auf einem Raspberry Pie angewendet werden, da es nicht mehr in den RAM passt. Auch auf Standart Laptops ist zu erwarten das ein größeres Modell echt zeit Berechnungen erschwert.

Um mit einer höheren Auflösung trainieren zu können wurden die nächsten Trainingseinheiten auf einem performateren Server durchgeführt. Der zusätzliche Arbeitsspeicher und die Grafikarten bringen bei einer aktivierung der CUDA beschleunigung einen grossen Geschwidikeitszuwachs .

Ein weiteres Problem das zu Tage trat war das Overfitting. Das Modell entwickelte sich gut auf Trainingsdaten, schnitt jedoch deutlich schlechter bei Testdaten ab, mit denen es nicht trainiert hatte. Die Kurven liefen bei mehr als 200 Epochen weit auseinander.

Weiterhin Rät [4] mehr Schichten benutzen, Beispielsweise abwechselnd eine Dichte schicht( dense layer) und eine Dropout Schicht.

Dropout Schichten im Neuronalen Netz verbesserten das Overfitting Problem. Somit waren mehr Trainingsläufe ohne overfitting möglich. Das neuronale Netz erreichte so werte von rund 85%.

Das Modell das TensorFlow bei voller Auflösung oder 307200 Eingangsneuronen erstellt ist über ein Gigabyte groß. Je größer das Modell ist, des do unwahrscheinlicher das ein Raspberry Pi das Modell in absehbarer Zeit auf ein Bild anwenden kann. Somit erwies sich der Ansatz einfach die Auflösung zu erhöhen als nicht zielführend für den Anwendungsfall.

Das Modell musste also kleiner werden ohne wesentliches Overfitting. Diese Ziehl wurde durch die Kombination von Dropoutschichten, mehr Durchläufen und kleinerer Pixelanzahl erreicht.

[4] Rät weiterhin, die Daten neu analysieren, wenn eine weitere Steigerung der Anzahl an Schichten und der Neuronen keine weitere Steigerung der Erkennungsrate bringt. Es wurden als Bilder mit sehr geringem Kontrast aussortiert, extrem Dunkle Bilder sowie Bilder auf denen der Marker sehr weit entfernt war. Sie wurden durch Bessere Daten ersetzt. (neue Bilder wurden angefertigt jeweils mit und Ohne Marker) Dieser Schritt verursachte eine Merkliche

Verbesserung des Trainingsergebnis.

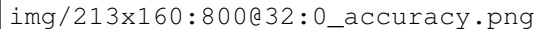


Abbildung 14. End ergebnis

Alle weiteren versucht das Model weiter zu verfeinern scheiterten. Die Ursache dafür ist nicht eindeutig geklärt. Eine mögliche Erklärung ist das der Datensatz für die komplexität des Models zu klein war . Auch möglich ist das die Grenzen der Optimierer ADAM und Stochastic Gradient Descent erreicht waren. Auch Programmierfehler konnten nicht ausgeschlossen werden.

### C. Anwendung des Trainingsmodells

//kommt noch

#### 1) Keras und Tensorflow auf Raspberry Pie Hardware:

Der initiale Plan sah vor, dass das neuronale Netz auf dem Raspberry Pi auszuwerten, damit der Roboter komplett autonom (ohne WLAN und externen Computer) fahren kann. Die Prozessorleistung und der RAM eines Raspberry Pies 3B+ ist zwar deutlich geringer als der eines Standard Laptops, aber die Nachteile eines verteilten Systems erschienen anfangs größer. So wirkte der architektonische Aufwand für verteilte Systeme und das Einarbeiten in die Bibliothek CV Bridge zum Versenden von Bildern mit ROS aufwendig.

Die meisten gängigen Bibliotheken und Frameworks sind unabhängig vom Zielbetriebssystem auf 64Bit Wortbreite ausgelegt, so auch Tensorflow und Keras. Das verwendete Ubuntu Image mit ROS für Raspberry Pi jedoch ist ein 32 Bit System. Das bedeutet, es gibt keine vorkompilierten Versionen von Tensorflow für das bisherige Setup. Das Kompilieren großer Tarfiles auf einem Raspberry Pi, dauert oft mehr als eine Stunde. Teilweise läuft der Raspberry Pi

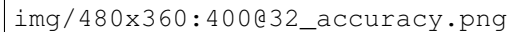


Abbildung 15. Kein sichtbarer Trainingserfolg

dabei so heiß, dass er abstürzt. Ein Ventilator erwies sich als hilfreich, um Abstürze während des Kompilierungsprozesses zu verhindern. Tensorflow ist eine große Bibliothek deren Kompilierung aufwendig ist. Sagt über den Raspberry Pi 4 : “The whole TensorFlow installation procedure from start to end takes many hours ( $\pm 33$  for Python,  $\pm 10$  for the C++ library). ”

Es ist anzunehmen, dass der Zeitaufwand für eine Kompilierung auf dem weniger Performanten Raspberry 3B länger dauert. Vor dem Hintergrund, dass es keine Garantie gibt, dass 64 Bit Libraries auf auf 32 Bit Systemen reibungslos funktionieren, ist eine Kompilierzeit von über einem Tag nicht akzeptabel.

Die Entscheidung für ein verteiltes System wurde getroffen. Dabei zeigten sich weitere erwähnenswerte Inkompatibilitäten. Für jede ROS Version (Distribution) existieren vor gepackte Installationspakete für ein Ubuntu Betriebssystem. Nur mit diesem System sind sie komplett kompatibel. Das Betriebssystem Ubuntu 18.04 ist mit ROS Melodic vorgesehen. Während das Betriebssystem Python 2.7 und Python 3.6.9. nutzt ist ROS Melodic auf Python 2.7 angewiesen. Die Python Versionen sind inkompatibel. Tensorflow2 benötigt eine aktuelle Python Version, die von ROS Melodic nicht ohne weiteres unterstützt wird.

Tensorflow1 ist zwar mit Python2.7 kompatibel, es ist jedoch nicht Threadsave. Das erschien anfangs kein Problem, da keine parallele Programmierung geplant war. Ein Prototyp-Programm mit Anwendung von neuronalem Netz lief problem-



---

```
Not found: Container localhost does not exist.
(Could not find resource: localhost/conv2d_2/kernel)
[[{{node conv2d_2/Conv2D/ReadVariableOp}}]]
```

---

Abbildung 16. Localost existiert nicht

---

```
InvalidArgumentError: Tensor
dropout__input:0, specified in either
feed_devices or fetch_devices
was not found in the Graph
```

---

Abbildung 17. Unverständliche Fehlermeldung

los auf Mac mit Python3. Dieses Programm sollte als Vorlage für einen ROS Knoten dienen. Auf Ubuntu in ROS traten jedoch viele kryptische Fehlermeldungen auf. Die Tatsache das Die Fehlermeldungen bei gleicher Eingabe so unterschiedlich waren verwirrte anfangs. ??, 17, 18

Es stellte sich heraus das entweder ROS, Python, Tensorflow oder die Kernel des Betriebssystems den Bildanalyse-Algorithmus als hochgradig parallelisierbar erkannt hat. Der Algorithmus im ROS Knoten nutzte mehr als einen Thread bei der Ausführung um die Mehrkern CPU auszunutzen. Die abschließende Ursache konnte zum zeit des Papers nicht geklärt werden. Laut Victor Meunier's Blog[Referenz hier] muss sichergestellt werden, dass jeder Thread seinen eigenen Graph und seinen eigene Session hat. Das Schlüsselwort "With" hilft den Kontext zu definieren. 19

Erst innerhalb dieses Blockes kann sicher auf das vortrainierte Modell zugegriffen werden, um damit eine Vorhersage zu machen.

2) *Algorithmus zur Bildanalyse :*  
// kommt noch

*D. Auswertung des Verhalten des Roboters*  
//kommt noch

## V. ERKENNTNISSE

Auch wenn dieses Projekt sehr lehrreich für alle Beteiligten war, kann ein studentisches Semesterprojekt nicht an die Leistungen im Rahmen von echten Forschungsarbeiten,

---

```
ValueError: Tensor Tensor
("activation_5/Softmax:0",
shape=(?, 2), dtype=float32)
is not an element of this graph.
```

---

Abbildung 18. Tensorflow Graphenelement unbekannt

---

```
with graph.as_default():
    with thread_session.as_default():
```

---

Abbildung 19. Verwendung von Initialisiertem Graph

durchgeführt von Wissenschaftlern mit Master oder Doktor Titel, heranreichen. Nicht nur das Budget sondern auch das Fachwissen, sowie die zeitlichen Gegebenheiten unterscheiden sich stark. Dieses Kapitel wird zuerst darauf auf mögliche Verbesserungen für eventuelle Nachfolgeprojekte eingehen, um dann einige Erkenntnisse zusammenzufassen.

### A. Empfehlungen für nachfolge Projekte

Empfehlungen auf Basis der Erfahrungen mit diesem Projekt sind, zunächst die Verbesserungsvorschläge für die Roboter Hardware und Software, danach Ansätze für die KI. Da dieses Projekt immer wieder Probleme mit der Inkompatibilität der veralteten Python Version und aktuellen Bibliotheken hatte, wird ein Hardware und Software Stack empfohlen, der durchgehend das aktuelle Python 3 verwendet. Auch die Leistungsgrenzen des Raspberry Pi waren schnell erreicht.

Empfehlungen:

- Besser ein Raspberry Pi 4\* als ein Raspberry Pi 3
- SD Karte mit mehr als 32GB
- Betriebssystem Ubuntu 20\* in 64 Bit Variante
- ROS Neotic\* auf allen Rechnern

\*oder aktueller wenn verfügbar

### B. Ansätze zur Verbesserung der Erkennung der Marker durch die K.I

Es ist zu überprüfen, ob ein Ecken erkennender Featurematchingalgorithmus nicht dem verwendeten Kantenerkennungsalgorithmus überlegen ist. Ansätze der künstlichen Intelligenz, die sich darauf konzentrieren *wo* die gesuchten Merkmalspunkte im Gesichtsfeld des Roboters ist, statt sich darauf zu beschränken *ob* ein Merkmalspunkt detektiert wurde, versprechen eine präzisere Navigation.

Es ist in Zukunft zu überprüfen, ob ein Training mit schwarz-weiß Bildern einen ähnlichen Trainingserfolg bringt. Bei gleicher Auflösung brauchen RGB Bilder drei mal so viele Daten wie schwarzweiß Bilder, da sie drei mal so viele Farbkanaäle brauchen. Das Umrechnen von Farbbildern in schwarzweiß Bilder ist dank Opencv keine rechenintensive Operation. Inwieweit zusätzliche Rechenoperationen die Serialisierungszeit, die Versandzeit und die Reserialisierung der Daten aufhebt sprengte den Rahmen dieser Arbeit. Inwiefern die Reduktion der Datenmenge das Gesamtsystem performanter macht ist noch zu überprüfen.

Des Weiteren ist darauf hinzuweisen, dass es schon viele gute Algorithmen zur Navigation mit Hilfe von optischen Markern gibt, deren Evaluation jedoch den zeitlichen Rahmen dieses Projektes gesprengt hätten. Für zukünftige Projekte ist ein Vergleich der bereits existierender Lösungen empfehlenswert.

### C. Fazit

//kommt noch

## LITERATUR

- [1] Francis X. Govers , Artificial Intelligence for Robotics, Packt Publishing ,2018
- [2] Amanda Decker , Bachelor Arbeit, Hochschule Mannheim ,2019
- [3] Learn Keras for Deep Neural Networks, Jojo Moolayil apress, 2019
- [4] <https://www.tamiya.com/english/rc/manuals.htm>
- [5] [https://blog.victormeunier.com/posts/keras\\_multithread/](https://blog.victormeunier.com/posts/keras_multithread/)

[1] [3] [5] [2] [4]

## LITERATUR

### *Fachliteratur*

- [2] Francis X Govers. *Artificial intelligence for robotics: Build intelligent robots that perform human tasks using AI techniques*. Packt Publishing Limited, 2018.
- [4] Jojo Moolayil, Jojo Moolayil und Suresh John. *Learn Keras for Deep Neural Networks*. Springer, 2019.

### *Web-Dokumente*

- [1] Amanda Decker. “Bildklassifikation auf einem Raspberry Pi Zero am Beispiel einer Ladestationserkennung”. Bachelor-Thesis. Hochschule Mannheim, 2019.
- [3] Radek Kozieł. *How to do multithreading with Keras*. 2019-02-10. URL: [https://blog.victormeunier.com/posts/keras%5C\\_multithread/](https://blog.victormeunier.com/posts/keras%5C_multithread/) (besucht am 18.07.2020).
- [5] TAMIYA. *RC Manual Download*. 2003. URL: <https://www.tamiya.com/english/rc/manuals.htm> (besucht am 18.07.2020).