



# Robotics **Rutesheim**

Dokumentation unserer Arbeit für den Future Engineers  
Wettbewerb der World Robot Olympiad.

Institution: Gymnasium Rutesheim

Teammitglieder: Maximilian Rath, Maximilian Schneider

# Inhalt

<b>Inhalt</b>	<b>2</b>
<b>Motorisierung</b>	<b>3</b>
Servo	3
Motorshield	3
Antrieb	3
<b>Energie &amp; Sensoren</b>	<b>4</b>
<b>Energie</b>	<b>4</b>
Stromversorgung	4
Tests/Probleme	4
Sensoren	4
Gyro-Sensor	5
Kamera	5
ToF-Sensor	5
<b>Hindernisse</b>	<b>6</b>
Neuronales Netzwerk	6
Motivation und Wahl des Systems	6
Aufbau des Netzwerks	6
Training des Netzwerks	7
Algorithmen zur Unterstützung des neuronalen Netzwerks	7
Navigation im Eröffnungsrennen	8
Navigation im Hindernisrennen	8
Herausforderungen bei der Programmierung	8
<b>Fotos</b>	<b>9</b>
<b>Engineering/Design</b>	<b>11</b>
Chassis	11
Anbauten	11
Umbauten	11
Probleme	11
<b>Anhang</b>	<b>12</b>
Trainings-Plot	12
Technische Zeichnungen	13

# Motorisierung

Im Themenbereich Motorisierung haben wir uns für zwei Motoren entschieden: einen Servomotor und einen Antriebsmotor.

## Servo

Der Servomotor, der für die Lenkung der Vorderachse zuständig ist, wird von 20 bis 80 Grad angesteuert. Dabei entsprechen 20 Grad in der Software der maximalen Lenkung nach links, 50 Grad der Fahrt geradeaus und 80 Grad die maximale Lenkung nach rechts.

## Motorshield

Der Antriebsmotor wird mit 12V betrieben. Diese kommen aus unserem 12V-Akku und werden am Motorshield gesteuert. Dabei haben wir uns für das Motorshield L293D entschieden, da wir mit diesem bereits im letzten Jahr gute Erfahrungen sammeln konnten. Dieses stellt außerdem, mit einem separaten 5V-Anschluss, die Spannungsquelle für unseren Servomotor bereit, damit wir die Anschlüsse des Raspberry Pi nicht überlasten.

## Antrieb

Der Motor greift direkt in ein Zahnrad ein, das wiederum mit einem weiteren, kleineren Zahnrad verbunden ist. Dies haben wir so gestaltet, um eine höhere Drehzahl auf unsere Räder übertragen zu können.

Die Übertragung von Zahnrad zu Rädern erfolgt über eine feste Hinterachse. Wir haben uns jedoch gegen die Verwendung eines Differentials entschieden, da dieses für unsere Kurvenfahrt mithilfe des Servos nicht notwendig ist und somit ausschließlich Platz verbraucht hätte. Außerdem wäre dies mit deutlich mehr Aufwand verbunden gewesen, denn wir hätten zunächst ein passendes Differenzial finden müssen.

Durch das direkte Zusammenspiel von Motor und Hinterachse bietet diese Antriebsart eine gute Traktion. Des Weiteren funktioniert sie sehr gut mit der durch den Servomotor gelenkten Vorderachse.

Wir hatten ebenfalls darüber nachgedacht, die gelenkte Achse nach hinten zu verlegen, also den Roboter „verkehrt herum“ fahren zu lassen, um ihn wendiger zu machen.

Wir haben uns dann jedoch dagegen entschieden, da in unserem Fall die Sensoren bereits früher in die neue Kurve gerichtet sind.

# Energie & Sensoren

## Energie

### Stromversorgung

Im Bereich der Energie haben wir verschiedene Möglichkeiten ausprobiert, uns dann allerdings für ein System mit zwei Stromquellen entschieden. Dabei haben wir zum einen eine Powerbank, welche den Raspberry-Pi mit 5V versorgt. Zusätzlich haben wir einen 12V Akku, welcher ursprünglich dem Betrieb von Elektronik Werkzeugen wie Akkuschaubern dient. Dieser bietet uns viele Vorteile: Die Bosch-Akkus waren bereits bei uns in der Schule vorhanden und wir hatten keine zusätzlichen Anschaffungskosten. Außerdem hatten wir im letzten Jahr schlechte Erfahrungen mit günstigen Akkus aus dem Internet gemacht, da diese schon nach kurzer Zeit nicht mehr auf die gewünschte Spannung kamen. Auf die Werkzeugakkus können wir uns erfahrungsgemäß verlassen und dies auch über einen langen Zeitraum. Außerdem wollten wir ein Akkusystem mit mehreren Akkus, um während Ladezeiten dennoch testen und fahren zu können, sowie im Falle eines Akku Defekts aber auch auf andere Akkus zurückgreifen zu können. Um die Akkus schnell wechseln zu können, sowie diese während der Fahrt gut verstauen zu können, haben wir uns dazu entschieden, eine Halterung 3D zu drucken, in welcher der Akku steckt. In diese haben wir Flachkontakte eingelassen, die perfekt in die Kontakte des Akkus passen.

### Tests/Probleme

Des Weiteren haben wir noch ausprobiert, das Motorshield nur mit einem 9V-Block zu betreiben, hier hat uns dann allerdings die Leistung für den Antriebsmotor gefehlt. Außerdem hatten wir noch versucht, auf die Powerbank zu verzichten und die 5V Ausgabe des Motor Shields als Spannungsquelle für den Raspberry-Pi zu nutzen. Dies hat allerdings nur kurzzeitig funktioniert, da das Motorshield unter der großen Last schnell überhitzt.

Um den Roboter vor Rennstart "scharf" stellen zu können, haben wir dann zwei Kippschalter eingebaut, mit welchen man beide Stromkreise separat schalten kann. Dabei kamen wir dann allerdings zu dem Problem, dass die Powerbank keine Spannung mehr für unseren Pi ausgab. Dies liegt daran, dass wir als Verbindungskabel ein reines Stromkabel nutzen. Moderne USB-Geräte und Powerbanks nutzen die Kommunikation über das Datenkabel D+ bzw. D-. Damit prüft das Gerät, ob die Spannungsquelle "sicher" ist, um Kurzschlüsse zu vermeiden. Um dieses Problem zu überbrücken, nutzen wir einen Adapter von USB-A auf USB-C. Dieser gibt der Powerbank das entsprechende Signal, dass sie Spannung freisetzen kann, gibt aber die volle Spannung weiter an das Stromkabel. Somit können wir den Pi trotzdem betreiben.

## Sensoren

In unserem Setup nutzen wir insgesamt 3 verschiedene Sensoren: Einen Gyro-Sensor, eine Kamera und 2 ToF-Sensor (Time-of-Flight-Sensor).

## Gyro-Sensor

Der Gyro-Sensor liefert uns wichtige Informationen für die Steuerung des Roboters. Der Z-Winkel, welchen wir durch den Gyro auslesen können, hilft unserem neuronalen Netzwerk nachzuvollziehen, wann die Kurve vollständig gefahren wurde. Außerdem können wir mit Hilfe des Z-Winkels feststellen, wann wir die 3 Runden vollständig gefahren sind. Wir haben bei der Platzierung des Sensors besonders darauf geachtet, dass er möglichst mittig und gleichzeitig stabil befestigt ist, um Fehler bei der Messung zu vermeiden. Wir haben uns für einen MPU6050 entschieden, da das Modul gut dokumentiert ist und wir es per I<sup>2</sup>C gut integrieren konnten.

## Kamera

Die Kamera liefert uns die wichtigsten Informationen für die Steuerung, da das neuronale Netzwerk den Großteil seiner Informationen über das Kamerabild bezieht, und die Kamera auch in diversen von uns genutzten Algorithmen eine zentrale Rolle spielt. Wir haben uns für ein „Raspberry Pi Cam Module 3“ entschieden, da es sich per Flexkabel simpel mit unserem Raspberry Pi verbinden lässt. Außerdem bietet sie uns mit gut dokumentierten Librarys die Möglichkeit, das Kamerabild sowohl als Datei zu speichern, als auch als Array auszulesen, was es praktischer für die Weiterverarbeitung macht.

## ToF-Sensor

Um Abstände zu den Barrieren rund um unseren Roboter festzustellen, nutzen wir einen ToF (Time-of-Flight) Sensor. Während wir im letzte Jahr noch Ultraschall-Sensoren für diese Aufgabe genutzt haben, haben wir uns diese Saison für ToF-Sensoren entschieden. Da diese Sensoren den Abstand per Laser feststellen, sind sie besonders, wenn der Roboter schräg an einer Wand steht, deutlich Fehlerunanfälliger. Außerdem vereinfacht die Ansteuerung der Sensoren über I<sup>2</sup>C das Auslesen und Einstellen der Sensoren. Wir haben uns dafür entschieden, nur zwei ToF-Sensoren zu verwenden, weil die von uns verwendeten Sensoren nur bis ca. 100cm korrekte und präzise Messergebnisse liefern. Aufgrund dessen, dass während des Großteils der Zeit der vordere Abstand vom Roboter zur Begrenzung vor ihm größer als diese maximale Distanz ist. Aus diesem Grund haben wir die ToF-Sensoren nur an der rechten und linken Seite des Vorbaus platziert.

# Hindernisse

Zur Steuerung unseres Roboters nutzen wir ein neuronales Netzwerk, das teilweise durch zusätzliche Algorithmen unterstützt wird. Im folgenden Kapitel werden das Konzept und die Umsetzung der Programmierung näher erläutert. Der vollständige Quellcode ist Open Source unter <https://github.com/maxc0des/future-engineers-25> verfügbar.

## Neuronales Netzwerk

### Motivation und Wahl des Systems

In der Brainstorming-Phase des diesjährigen Wettbewerbs haben wir festgestellt, dass viele für die Steuerung wichtige Informationen visuell – also mit Hilfe einer Kamera – erfasst werden können. Da sich solche visuellen Daten jedoch nur schwer über klassische Algorithmen verarbeiten lassen, haben wir uns entschieden, ein neuronales Netzwerk für einen Großteil der Steuerung einzusetzen.

Wir haben uns für Python als Programmiersprache entschieden, da es einerseits nativ auf dem Raspberry Pi läuft und auch für Machine Learning optimal geeignet ist.

### Aufbau des Netzwerks

Das neuronale Netzwerk haben wir mit dem PyTorch-Framework entwickelt. Es handelt sich um ein Convolutional Neural Network (CNN), bestehend aus sieben Blöcken von Convolutional Layers, einem adaptiven Pooling-Layer sowie einem Block aus Fully Connected Layers. Diese Architektur hat sich in unseren Tests als besonders geeignet erwiesen, da sie genügend Tiefe besitzt, um wichtige Daten-Merkmale zu erfassen, gleichzeitig jedoch nicht zu tief ist, um auch auf unserem Raspberry Pi noch reibungslos arbeiten zu können.

Das Netzwerk verarbeitet ein Bild (welches auf die relevanten Bereiche zugeschnitten ist) sowie die Daten zweier ToF-Sensoren und des Gyro-Sensors. Diese beiden Datentypen werden in einem Tensor der Größe 128x128 mit sechs Kanälen zusammengeführt – drei Bildkanäle für die jeweiligen RGB-Farbwerte, je ein Kanal pro Abstandswert des ToF-Sensor und einen Kanal für den Z-Winkel des Gyro-Sensor. Das Netzwerk gibt daraus einen Lenkwinkel aus.

```
#zusammenführung der daten vor vorhersage
gyro = get_gyro("gyro")
    tof = list(get_tof())
    image = Image.fromarray(img)
    image = crop_image(image)
    image = transforms.Resize((128, 128))(image) #might change size
    image = transforms.ToTensor()(image)
    image = transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))(image)

    gyro = torch.tensor([
        gyro
    ], dtype=torch.float32)
```

```

    tof = torch.tensor([
        tof[0],
        tof[1]
    ], dtype=torch.float32)

    gyro_expanded = gyro.view(1, 1, 1).expand(1, 128, 128)
    tof_expanded = tof.view(2, 1, 1).expand(2, 128, 128)
    combined_input = torch.cat((image, tof_expanded, gyro_expanded),
dim=0)

```

## Training des Netzwerks

Trainiert wurde das Modell mit Daten aus eigenen Fahrten, bei denen wir den Roboter manuell per Controller gesteuert und dabei gleichzeitig Kamerabilder, ToF-Daten, Gyro-Daten und Lenkwinkel aufgezeichnet haben. Vor dem Training werden die Daten vorverarbeitet: Reihen mit fehlenden Werten werden entfernt, und die Bilder werden auf die relevanten Bereiche zugeschnitten.

Zur Data Augmentation verändern wir bei zufällig ausgewählten Bildern die Helligkeit. Dadurch vergrößern wir nicht nur unser Dataset, sondern machen das Netzwerk robuster gegenüber variierenden Lichtverhältnissen.

Während des Trainings visualisieren wir sowohl den Loss als auch den Test-Loss über einen Plot (Siehe Anhang). Zudem verwenden wir Early Stopping, um das Training rechtzeitig zu beenden, bevor Overfitting einsetzt (siehe Codeausschnitt).

```

def earllystop(values: list):
    global patience_counter, window_size
    if len(values) > min_epochs:
        derivatives = np.diff(values[-window_size:])
        print(f"Avg Derivatives: {np.mean(derivatives)}")

        if np.mean(derivatives) > 0: # Wenn der Test-Loss steigt
            patience_counter += 1
            print(f"Patience: {patience_counter}/{patience}")
        else:
            patience_counter = 0

        return patience_counter >= patience
    return False

```

## Algorithmen zur Unterstützung des neuronalen Netzwerks

Während der ersten Tests zeigte sich, dass unser neuronales Netzwerk nicht alle Aufgaben zuverlässig lösen kann. Daher haben wir entschieden, es durch klassische Algorithmen in bestimmten Situationen zu ergänzen. Insgesamt nutzen wir drei unterstützende Algorithmen:

1. Fahrtrichtungserkennung:

Zu Beginn eines jeden Rennens bestimmt der Roboter die Fahrtrichtung, indem er

die Abstände zu den Banden in der ersten Kurve vergleicht.

2. Stillstandserkennung und Rücksetzen:

Während des Rennens vergleichen wir das aktuelle Kamerabild mit dem vorherigen. Bleibt das Bild nahezu unverändert, ist der Roboter vermutlich festgefahren. In diesem Fall führen wir ein kurzes Zurücksetzen durch und übergeben die Steuerung anschließend wieder an das neuronale Netzwerk.

3. Ausrichtung und Farberkennung:

Im Hindernisrennen wird am Anfang jeder Geraden ein Algorithmus eingesetzt, um den Roboter zuverlässig auszurichten und die Farbe des nächsten Hindernisses zu erkennen. Mithilfe des Gyroskops pendelt sich der Roboter durch wiederholtes Vor- und Zurückfahren ein. Dabei wird der Lenkwinkel dynamisch abhängig vom aktuellen Gyro-Winkel angepasst. Zusätzlich berücksichtigen wir die Distanz zur äußeren Bande, um ein Anstoßen zu vermeiden und möglichst mittig auf der Bahn zu starten. Außerdem stellen wir die Farbe des Hindernis auf der nächsten Geraden fest und geben sie an den Hauptalgorithmus, in dem das neuronale Netzwerk läuft, zurück.

## Navigation im Eröffnungsrennen

Das Eröffnungsrennen beginnt der Roboter mit der Feststellung der Fahrtrichtung. Dazu fährt er zur ersten Kurve und vergleicht die Abstände nach rechts und links, um zu ermitteln, ob die Strecke im oder gegen den Uhrzeigersinn befahren wird.

Anschließend wird ein richtungs spezifisches neuronales Netzwerk geladen. Während der Entwicklung haben wir festgestellt, dass kleinere, spezialisierte Netzwerke deutlich bessere Ergebnisse liefern als ein einzelnes universelles Modell.

Nach der Auswahl übernimmt das Netzwerk die Steuerung des Roboters.

## Navigation im Hindernisrennen

Genau wie das Eröffnungsrennen, beginnt auch das Hindernisrennen mit der Bestimmung der Fahrtrichtung. Auf den Geraden wird der Roboter durch ein neuronales Netzwerk gesteuert. In den Kurven setzen wir jedoch den vorher beschriebenen Algorithmus zur Ausrichtung und Farbbestimmung ein. Auch hier nutzen wir für jede Kombination aus Fahrtrichtung und Hindernis Farbe ein eigenes spezialisiertes Netzwerk.

## Herausforderungen bei der Programmierung

Während der Testläufe traten wiederholt Probleme mit der Leistung unserer Antriebsmotoren auf. Ab einem bestimmten Lenkwinkel benötigten die Motoren deutlich mehr Strom, um den Roboter mit konstanter Geschwindigkeit zu bewegen.

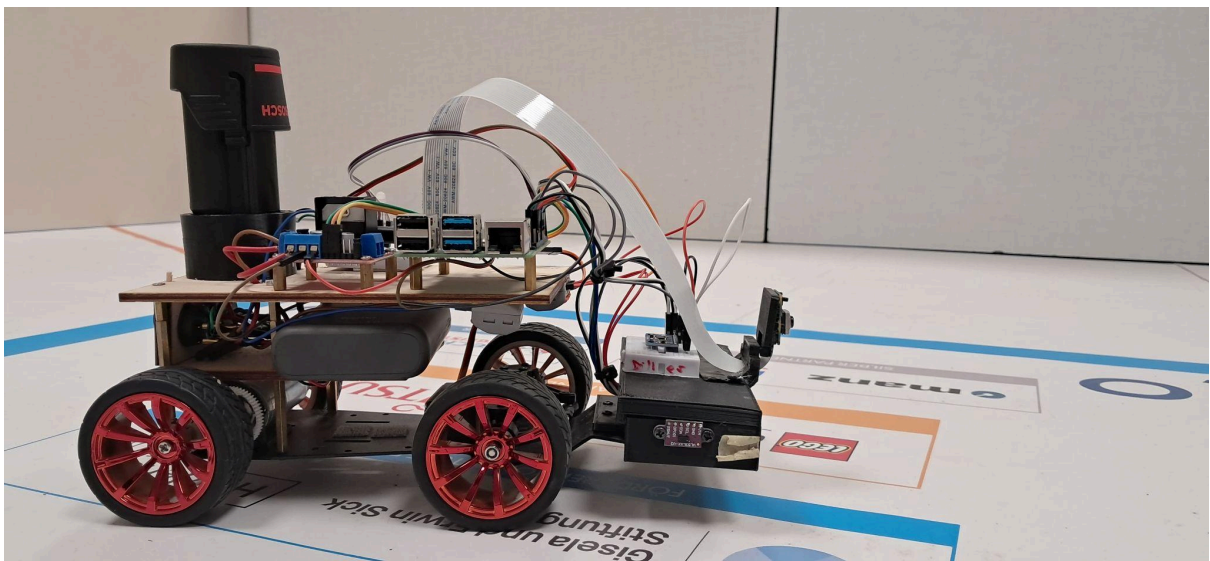
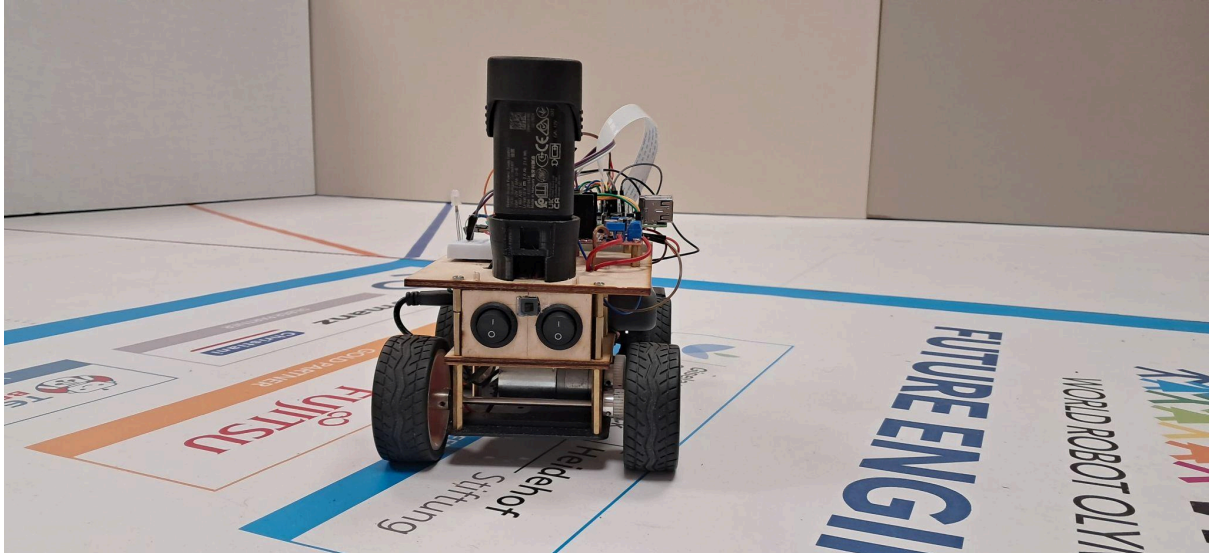
Daher haben wir die Geschwindigkeit der Motoren dynamisch angepasst: Zusätzlich zum Basiswert („basic\_speed“) wird abhängig vom Lenkwinkel ein anteiliger „speed\_boost“ hinzugefügt. So stellen wir sicher, dass der Roboter auch in Kurven stabil fährt.

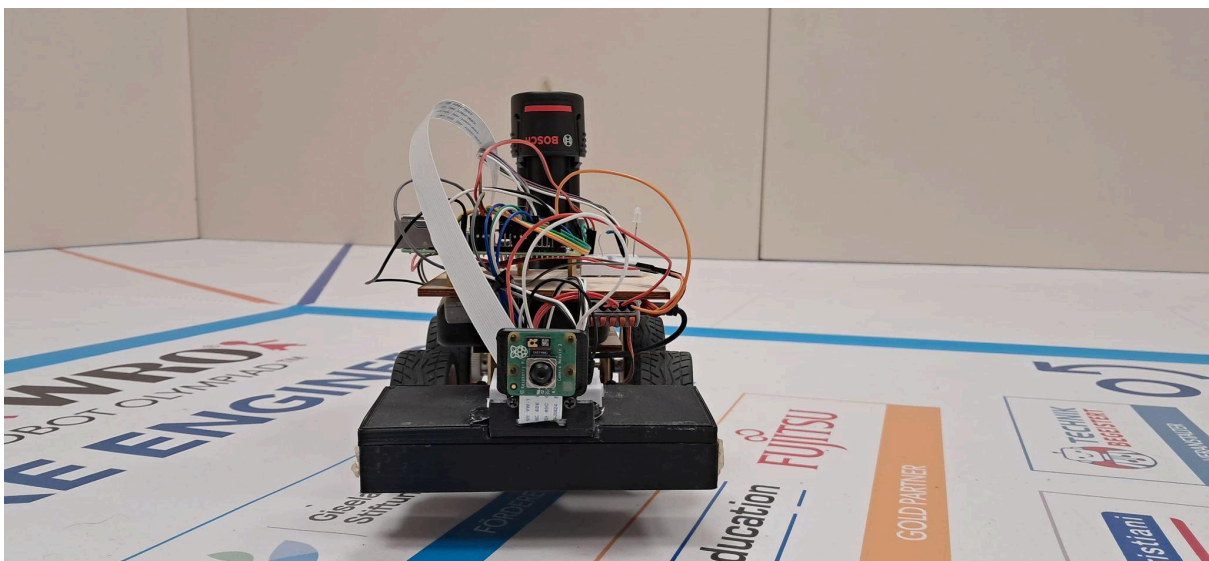
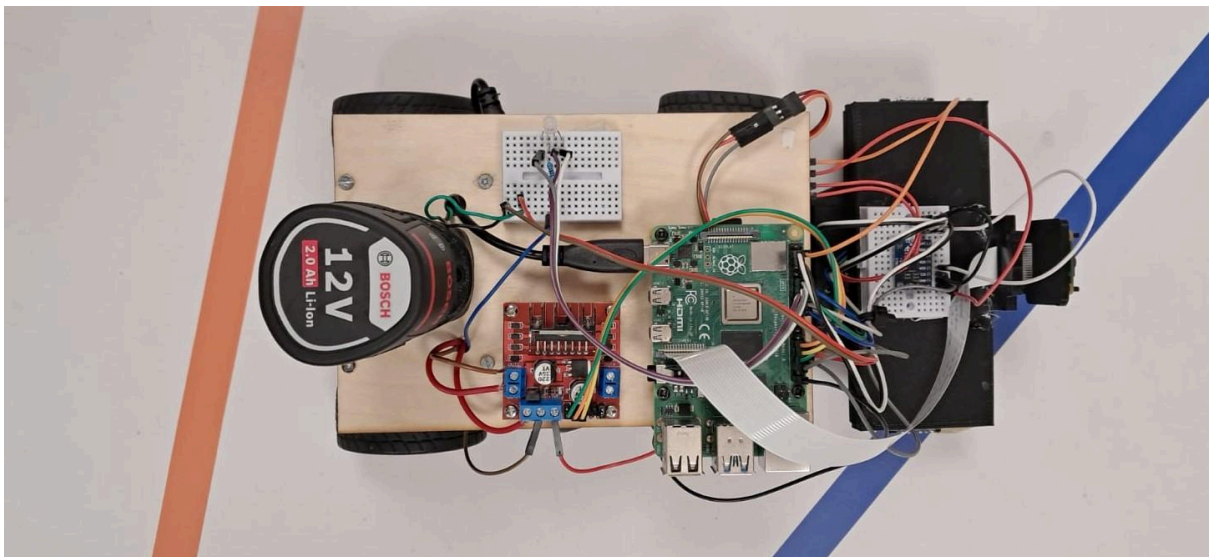
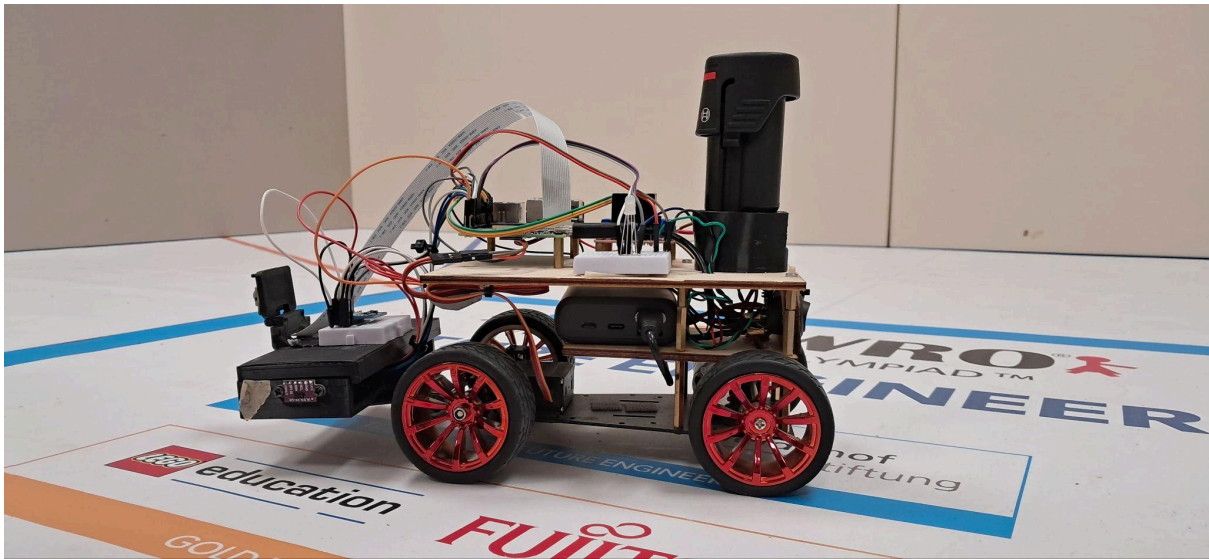
```
#speed variabel anpassen
divisor = 30.0
additional_speed = (abs(steering - 50) / divisor) * speed_boost
```



```
speed = basic_speed + additional_speed
```

## Fotos





# Engineering/Design

Auch dieses Jahr verwenden wir wieder das von der WRO gesponserte Funduino-Set, allerdings haben wir dieses modifiziert und stellenweise abgeändert. Allgemein verwenden wir einige Designs und Ideen aus dem Vorjahr, welche sich bewährt haben. Andere haben wir weiterentwickelt und optimiert.

## Chassis

Als Chassis verwenden wir das 4WD Chassis mit Servo-Lenkachse von Funduino. Allerdings nur die Edelstahl Grundplatte mit Motoren, Servo und Räder, jedoch nicht die mitgelieferten Acrylplatten. Das Chassis bietet uns eine stabile und praktische Grundlage, welche sich auch letztes Jahr schon bewährt hat.

## Anbauten

Um noch zusätzlichen Platz zu gewinnen, sowie eine bestmögliche Position für unsere ToF-Sensoren sowie die Kamera zu gewähren, haben wir an der Front noch eine Erweiterung 3D-gedruckt (siehe technische Zeichnungen im Anhang). Diese ist leicht, stabil und einfach konstruiert. Selbige haben wir dann an Stelle des Schaumstoffblocks befestigt und sowohl zwei ToF-Sensoren sowie die Kamera befestigt (siehe Fotos). Die beiden ToF-Sensoren haben wir dafür seitlich eingelassen, damit sich diese möglichst weit unten befinden. Die Kamera haben wir mit einem separaten Winkel montiert, damit diese für den Transport besser abnehmbar ist und gegebenenfalls noch der Kamerawinkel verändert werden kann.

## Umbauten

Anstatt der Acrylplatten haben wir dieses Jahr 4mm dicke Leimholzplatten ausgelasert. Bei diesen konnten wir mittels CAD die Position für Motorshield, Raspberry-Pi, Steckbrett und die Akkuhalterung. Dabei haben wir zwei „Stockwerke“ übereinander. Am unteren ist die Powerbank für den Raspberry-Pi sowie die Startschalter montiert. Auf dem oberen befinden sich die anderen Bauteile. Dabei haben wir den Raspberry-Pi möglichst weit vorne platziert, damit die Kabelstrecken zur Kamera und den Abstandssensoren nicht zu lang werden. Der Akku ist hinten platziert, um den Schwerpunkt des Fahrzeugs in Richtung der angetriebenen Achse zu bekommen.

## Probleme

Anfangs hatten wir noch keine ordentliche Befestigung für den Akku, da dieser nur mittels zwei Kabel mit Flachkontakten angeschlossen war. Dann haben wir nach etwas Recherche eine Datei gefunden und diese noch etwas angepasst, um den Akku stabil einstecken zu können. Außerdem mussten wir den Vorderanbau mehrfach ausdrucken, bis alle Steckteile gut ineinander passen.

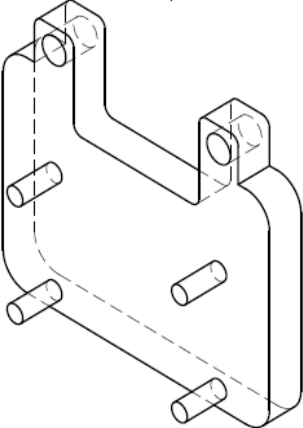
# Anhang

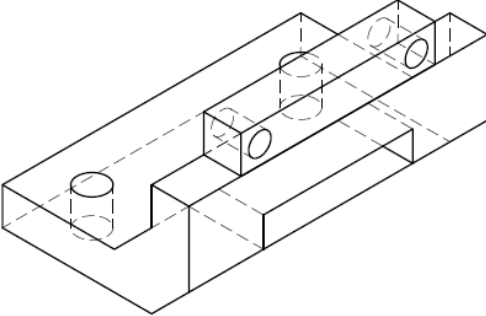
## Trainings-Plot

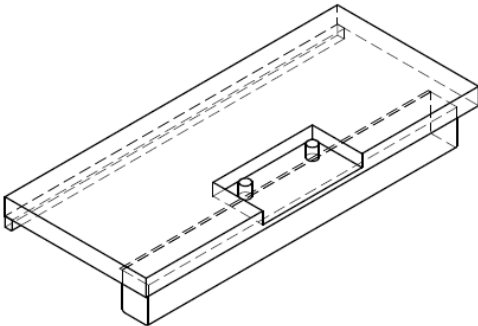


Plot des Train- und Test-Loss im Verlauf der Trainigsepochen

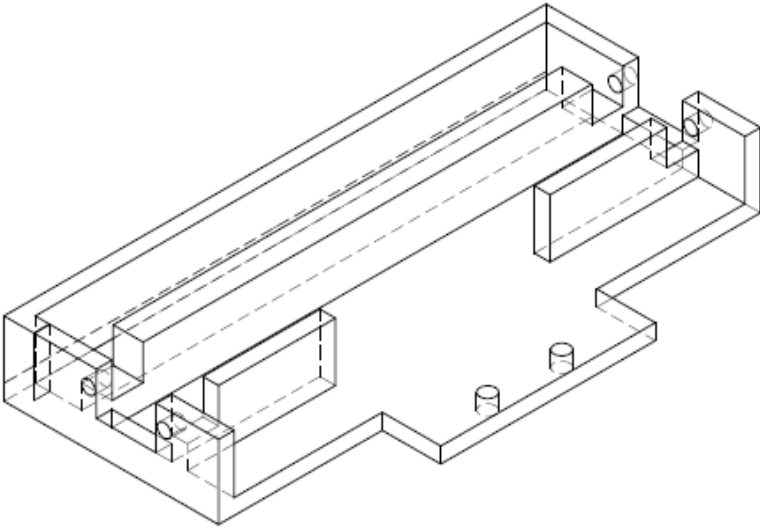
## Technische Zeichnungen

	1	2	3	4													
A					A												
B					B												
C					C												
D	<table border="1"> <tr> <td>Dept.</td><td>Technical reference</td><td>Created by <b>Maximilian Rath 22.02.2025</b></td><td>Approved by</td></tr> <tr> <td colspan="2" rowspan="2"></td><td>Document type</td><td>Document status</td></tr> <tr> <td colspan="2">Title <b>Kamera Aufnahme</b></td></tr> <tr> <td colspan="2"></td><td>Rev.</td><td>Date of issue</td><td>Sheet <b>1/1</b></td></tr> </table>	Dept.	Technical reference	Created by <b>Maximilian Rath 22.02.2025</b>	Approved by			Document type	Document status	Title <b>Kamera Aufnahme</b>				Rev.	Date of issue	Sheet <b>1/1</b>	D
Dept.	Technical reference	Created by <b>Maximilian Rath 22.02.2025</b>	Approved by														
		Document type	Document status														
		Title <b>Kamera Aufnahme</b>															
		Rev.	Date of issue	Sheet <b>1/1</b>													
	1	2	3	4													

	1	2	3	4													
A					A												
B					B												
C					C												
D	<table border="1"> <tr> <td>Dept.</td><td>Technical reference</td><td>Created by <b>Maximilian Rath 22.02.2025</b></td><td>Approved by</td></tr> <tr> <td colspan="2" rowspan="2"></td><td>Document type</td><td>Document status</td></tr> <tr> <td colspan="2">Title <b>Winkel für Kamera</b></td></tr> <tr> <td colspan="2"></td><td>Rev.</td><td>Date of issue</td><td>Sheet <b>1/1</b></td></tr> </table>	Dept.	Technical reference	Created by <b>Maximilian Rath 22.02.2025</b>	Approved by			Document type	Document status	Title <b>Winkel für Kamera</b>				Rev.	Date of issue	Sheet <b>1/1</b>	D
Dept.	Technical reference	Created by <b>Maximilian Rath 22.02.2025</b>	Approved by														
		Document type	Document status														
		Title <b>Winkel für Kamera</b>															
		Rev.	Date of issue	Sheet <b>1/1</b>													
	1	2	3	4													

	1	2	3	4	
A					A
B					B
C					C
D		Dept.	Technical reference	Created by Maximilian Rath 22.02.2025	Approved by
		Document type		Document status	
		Title Roboter_Anbau_Deckel		DWG No.	
		Rev.	Date of issue	Sheet 1/1	
	1	2	3	4	

					
Dept.	Technical reference	Created by Maximilian Rath 20.02.2025	Approved by		
		Document type	Document status		
		Title Roboter_Vorderanbau		DWG No.	
		Rev.	Date of issue	Sheet 1/1	