

DOKUMENTATION

Future Engineers 2024

robofactory (Jesse Born, Julian von Hoff)



April 2024

Eingereicht bei der WRO Regio in Waldkirch bei Freiburg

1 MOTORIZIERUNG

Als Hauptantrieb verwenden wir eine Handelsübliche Kombination aus Fahrtenregler und Motor, die aus dem Modelbau stammt. Wir verwenden einen 21.5T gewickelten Motor von Tamiya mit eingebautem Drehgeber und das dazugehörige ESC TBLE-04SR. Über ein Differentialgetriebe und eine gesamthaftes Untersetzungsverhältnis von 3.122:1 bringt unser Fahrzeug seine Motorleistung über einen Heckantrieb auf den Boden. Unsere Räder bestehen aus Felgen die im Lieferumfang unseres Chassis dabei waren, so wie 56x28 mm Reifen von LEGO.

2 ENERGIE & SENSOREN

Der einzige Sensor den wir verbaut haben, ist eine RGB-D Kamera von Orbbec, eine Astra Embedded S. Dieses verhältnismäig preiswerte Auslaufprodukt ermöglicht nebst der klassischen Webcamfunktion auch das erfassen von Tiefeninformation mittels strukturiertem Infrarotlicht. Via USB / UVC-Protokoll wird das Kamerabild an unseren Hauptrechner, einen Raspberry Pi 4B übertragen.

Die Stromversorgung erfolgt über einen sechszelligen NiMh-Akku. Um den Raspberry Pi mit Strom zu versorgen, wird ein BEC (Battery Eliminator Circuit) aus dem Modellbau verwendet. Mit den Pin 2 und 6 des GPIO-Headers des Pi, wird dieser schlussendlich mit Strom versorgt. Genau dieses technische Detail bereitete uns am meisten Kopfschmerzen, sowie ein totes Raspberry Pi. Wir konnten jedoch nicht genau feststellen, weshalb der Pi seinen "magischen Rauch" freigab.

3 HINDERNISSE

Das von unserer Kamera erfasst RGB- bzw. BGR-Bild folgt nun 2 Pfaden: Der erste ist dafür zuständig, während dem Hindernissrennen die farbigen Hindernisse zu erkennen. Entlang dem zweiten Pfad werden mittels Graustufenbild und Hough-Lines-Algorithmus die Banden des Spielfeldes erkannt und auf einem virtuellen Spielfeld lokalisiert.

3.1 BANDEN

Als erstes erstellen wir ein Bild, in dem nur alle (fast) schwarzen Pixel abgebildet sind. Darin suchen wir anschliessend nach kontrastreichen Kanten. Haben wir die Kanten gefunden, segmentieren wir mittels probabilistischer Hough-Transformation zusammenhängende Kanten aus. Nun filtern wir diese Kanten weiter, wir überspringen alle Linien, welche...

1. keinen Endpunkt in einem Streifen um die Bildmitte haben
2. eine Nulllänge haben
3. komplett über der Bildmitte liegen
4. vertikal im Bild stehen

So erhalten wir pro Wand eine Linie, an der unteren Kante dieser, an der Stelle wo sie auf die Spielfeldmatte trifft. Mittels des 2. Strahlensatzes können wir nun den Abstand schätzen.

$$d = \frac{1}{(y_{Endpunkt} - y_{Bildmitte})} * s_d$$

(d : Abstand, $y_{Endpunkt}$: y-Koordinate eines Linienendpunktes im Bild, $y_{Bildmitte}$: Parallaxenmitte des Kamerabildes und s_d : konstanter Faktor, der sich zwar aus den intrinsischen Kameraparametern ableitet, jedoch experimentell bestimmt wurde.)

Wände werden nun in drei Klassen unterteilt:

- Rechts: Eine Wand am rechten Bildrand
- Links: Eine Wand am linken Bildrand
- Mitte: Eine Wand, deren Steigungswinkel unter 0.05 (rad) liegt.

Diese Klassifizierung wird auch zu Beginn des Laufs zur Feststellung der Rundenrichtung verwendet
– ist eine linke Wand sichtbar, wird im Uhrzeigersinn gefahren.



```
def findWallLines(edgesImg):
    lines = cv2.HoughLinesP(edgesImg, 1, np.pi/360,
                           threshold = houghparams['threshold'],
                           minLineLength = houghparams['minLineLength'],
                           maxLineGap = houghparams['maxLineGap'])

    if lines is not None:
        lines = list(lines)
    else:
        lines = []
    def lineSort(line):
        return line[0][0]
    lines.sort(key=lineSort)

    filteredLines = []
    for line in lines:
        x1, y1, x2, y2 = line[0]
        if y1 == 0 or y2 == 0:
            continue
        pass
        if abs(y1 - centerheight) > centerstripheight and abs(y2 - centerheight) >
centerstripheight:
            continue
        pass
        if y1 < centerheight or y2 < centerheight:
            continue
        pass
        if abs(math.atan2(y2-y1, x2-x1)) > math.pi/3:
            continue
        pass
        filteredLines.append([x1 + depthOffsetX, y1 + depthOffsetY, x2 + depthOffsetX, y2 +
depthOffsetY])
    return filteredLines
```

3.2 STARTRICHTUNG

Zu Beginn des Laufs ist nur jeweils eine mittlere Wand, sowie eine rechte oder linke Wand sichtbar. Ist eine linke Wand sichtbar, ist die Umlaufrichtung für diese Runde im Uhrzeigersinn. Ist es eine rechte Wand, im Gegenuhzeigersinn.

3.3 FARBENE HINDERNISSE

Das originale RGB-Bild wird in den HSV-Farbraum umgerechnet, um rot und grün deutlich unterscheiden zu können. Diesen Trick kennen und nutzen wir bereits seit mehreren Jahren in der

Kategorie RoboMission der WRO. Pixel die nun zwischen der Rot- bzw. Grüngrenzen liegen, werden als der Farbe entsprechend gespeichert. Auf diesen Binärbildern werden nun die Konturen erkannt und die Mittelpunkte sowie die Breite und Höhe bestimmen.

Analog zu oben, jedoch nur mit der Höhe der erkannten Kontur, wird mittels Strahlensatz der Abstand zum Hinderniss geschätzt.

3.4 STEUERSIGNAL

Mit dem Abstand gewichtetet, wird nun für jedes erkannte Objekt, also Banden und Hindernisse, dessen bevorzugte Richtung in einen Wert summiert.

$$s := \frac{\sum \frac{1}{d_i}}{T}$$

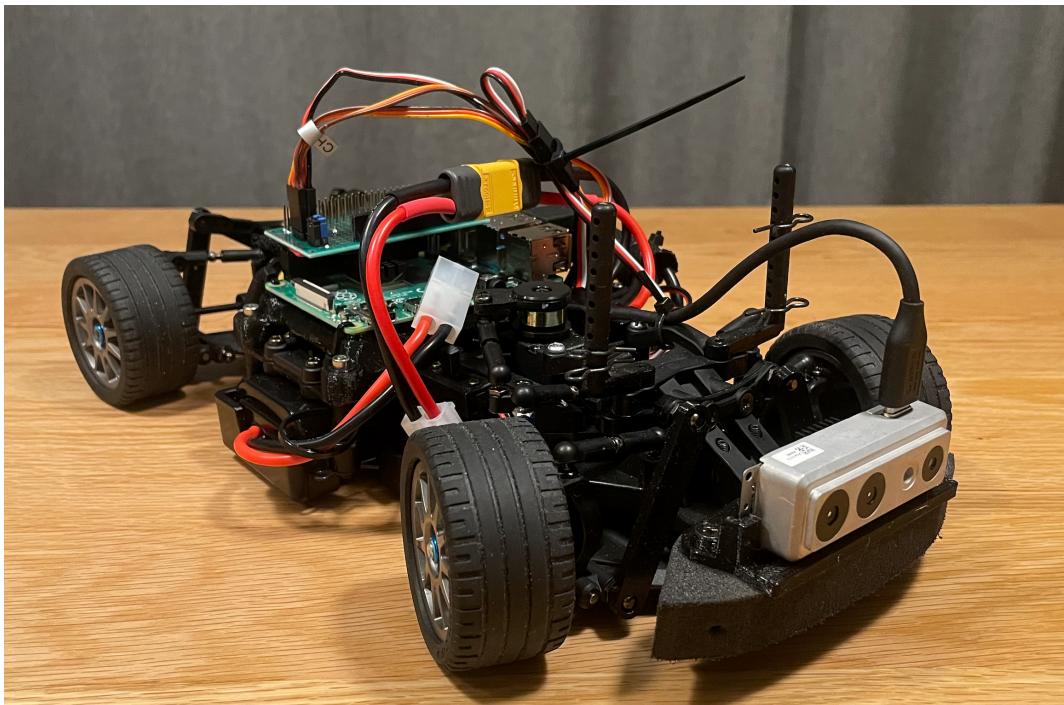
Dieser Wert wird dann normalisiert auf das geschlossene Intervall $[-1; 1]$. Der normalisierte Wert wird dann auf den Lenkanschlag des Servos abgebildet, so dass 0 auch einem gerade getrimmten Auto entspricht.

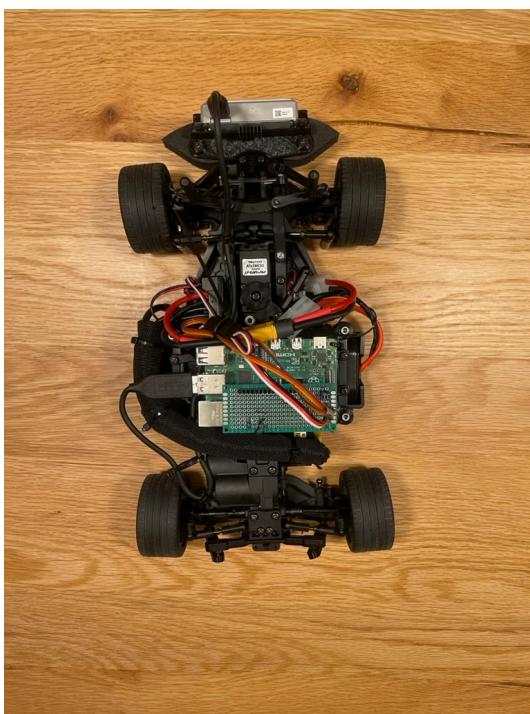
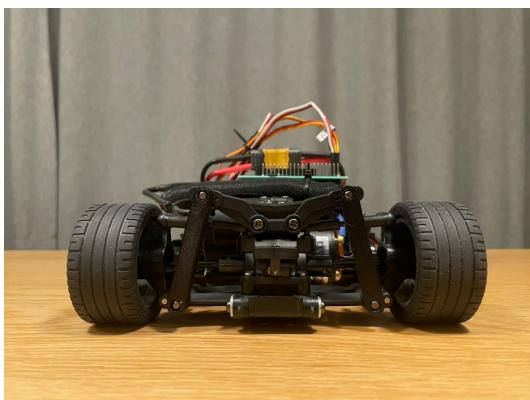
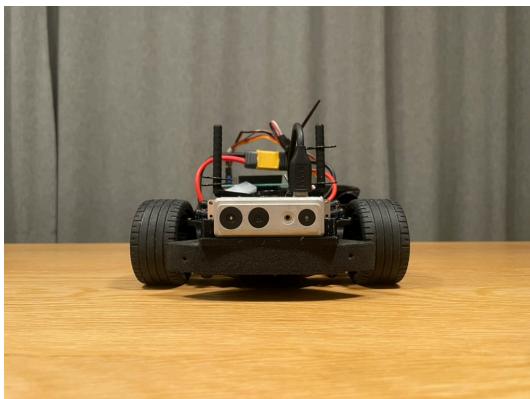
$$\alpha := [\frac{s}{16.0}]_{-1}^1$$

3.5 ENDE DER RUNDE

Um am Ende der Runde im Startbereich stoppen zu können, zählen wir wie oft wir um einen Ecken abbiegen. Dies funktioniert, da während dem Umfahren eines Ecken keine mittleren Wände erkannt werden. Wir während mehr als ca. 600ms keine mittlere Wand erkannt, zählen wir um eins hoch. Erreicht der Zähler ≥ 12 und das Auto sieht eine mittlere Wand $\leq 1600\text{mm}$ vor sich, beendet es den Lauf automatisch.

4 FOTOS





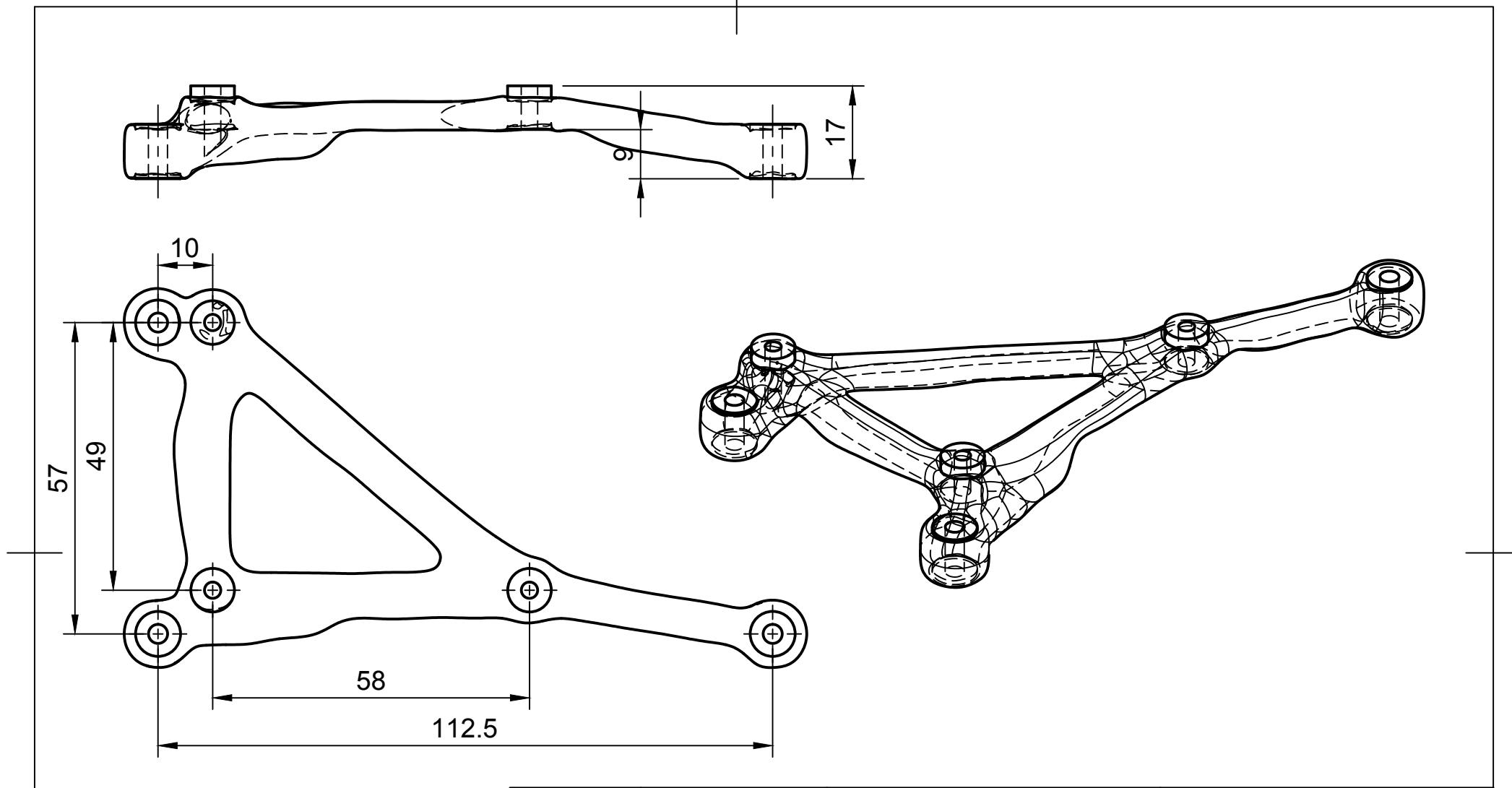
5 ENGINEERING & DESIGN

Der Bausatz des Chassis stammt ebenfalls aus dem Hause Tamiya – den klassiker M-08 Concept haben wir modifiziert, dass der Einschlag der Lenkung grösser und die Federung so steif wie nur möglich ist. (Siehe Kapitel ”3 - Hindernisse“). Die vordere Stossstange, bestehend aus einem dichten Schaumstoff haben wir an der Bandsäge gekürzt, so dass die gesamtlänge des Autos auch in die vorgegebenen 300 mm passt. Ergänzt haben wir den fahrbaren Unterbau mit einer Kamerahalterung über der vorderen Stossstange. Die Kamera hält darin Reibungsbasiert und lässt sich für Wartungsarbeiten rund um diese leicht entfernen. Aus Notdurft – um unseren Wanderkennsalgorithmus zu unterstützen – haben wir die hydraulische Federung des Chassisbausatzes mit soliden, 3d-gedruckten Teilen ersetzt. Damit vermeiden wir, das unser Auto rollt oder nickt – denn dies macht die visuelle Distanzschätzung ungenau bis unbrauchbar.

Um unsere Recheneinheit auf dem Auto zu befestigen, haben wir ebenfalls eine Halterung designet und 3D-gedruckt. (Siehe Anhang)

6 ANHÄNGE

- Konstruktionszeichnung Raspberry-Pi-Halterung
- Konstruktionszeichnung Kamerahalterung



Dept.	Technical reference	Created by Jesse Born	Approved by
		29.04.24	
	Document type	Document status	
	Title Raspberry PI mount	DWG No.	
Rev.	Date of issue	Sheet	1/1

