



## Team Nibot

Vincent Menzel (5855524)

Marius Lindt (5446728)

Enrico Bachus (7605326)

## Beschreibung des Projektablaufs & des Fahrzeugs:

Zunächst wurden durch den Dozenten die Möglichkeiten beim Erstellen eines Roboters dargestellt. Diese wurden im Anschluss innerhalb des Teams besprochen. Dabei konnte sich auf die Standardausstattung plus einem zusätzlichen OLED Screen für Visualisierungen und einen Lautsprecher für Audio Effekte (wie z.B. Sirenen) festgelegt werden. Da ein Prototyp mit Standardausstattung bereits vorhanden war wurde sich aufgrund der fehlenden Erfahrung im Bereich der Elektrotechnik dafür entschieden diesen zu übernehmen und zu mit den beiden weiteren Elementen zu erweitern. Im nächsten Schritt wurde eine Teststrecke mit allen zu absolvierenden Strecken Elementen aufgebaut und der Standardcode, der zur Orientierung über das LMS Moodle bereitgestellt wurde, mit Hilfe der Strecke optimiert.

### Erste Schritte mit dem Roboter:

Zunächst wurde der serielle Micro B USB Treiber installiert und der serielle Port am PC mit Portnummer und Speed (115200) eingerichtet. Anschließend wurde der Putty zum seriellen Verbindungsaufbau mit dem Roboter (ADAFRUIT Metro M4) verwendet um den Console Output des Roboters zu sehen. Dabei konnten die Sensorwerte (print Statements des Programmcodes) ausgelesen und der Programmcode durch das Abfahren der Teststrecke optimiert werden.

Abweichend vom Default Code wurden die beiden äußeren Sensoren aktiviert und bei der Entscheidungsfindung berücksichtigt. Erkennen diese die schwarze Linie so wird eine scharfe Kurvenfahrt ausgelöst. Weiterhin wurde die Ausnahme, *dass er mit den zuletzt verwendeten Werten weiter fahren soll* implementiert, falls

keiner der Sensoren eine schwarze Linie erkennt. Dadurch wird sichergestellt, dass der Roboter wieder zurück zu der schwarzen Linie findet. Um das Schlangenlinien-Fahren zu vermeiden, wurde, falls nur die inneren Sensoren die Linie erkennen, die Motorgeschwindigkeit lediglich



Abbildung 1: Teststrecke

reduziert, statt vollständig abzubremsen. So fährt der Roboter dann bloß eine leichte Kurve, statt einer Steilkurve.

### **Fine Tuning des Roboter Codes:**

Nachdem der grundlegende Code geschrieben war, ging es an das Optimieren des Roboter Codes. Dabei wurden der mögliche Fehler *“Überschreitung der Minimal und Maximalwerte der Sensoren”* durch die Verwendung des maximalen Wertes bei der Überschreitung und der Verwendung des minimalen Wertes bei Unterschreitung abgefangen. Nach diesem Prinzip wurden auch ähnliche Ausnahmen abgefangen. Mehr dazu in der detaillierten Code Beschreibung.

### **Das besondere Merkmal des Nibot Roboters:**

Um den Roboter des Teams Nibot besonders hervorzuheben wurde der Roboter mit einem OLED Display versehen, mithilfe dessen der aktuelle Speed des linken und rechten Reifens angezeigt werden soll. Dazu wird die Richtung angezeigt, in die der Roboter gerade fahren möchte, was auch beim Debugging hilft. Ein prävalentes Problem, dass bei der Implementierung des Displays auftrat, war die alte, bzw. leistungsschwache Hardware des adafruit Metro M4. Einige Libraries konnten hierfür nicht genutzt werden, wodurch das Team bei einer Vielzahl von Tutorials schließlich nicht mehr folgen konnte. Auch Animationen oder ähnliches wurden nicht unterstützt. Die Adafruit IDE konnte das Board ebenfalls nicht erkennen. So musste das Team an vielen Stellen Workarounds finden und ggf. auf einige Zusatzfeatures verzichten.

Ferner ist dem Team aufgefallen, dass eine Datei für die Darstellung von Schrift nicht in der aktuell installierten Version von Circuit Python fehlte. Durch langes Recherchieren gelangten wir auf ein Github Issue, das genau dieses Problem ansprach und fehlende Dateien zu Verfügung stellte. Waren diese installiert, funktionierte schließlich der Text Output am OLED Display. Danach wurden die Geschwindigkeitswerte der Reifen und die Darstellung der Richtung in der Anzeige implementiert. Als besonderes I-Tüpfelchen wurde der Song-Text von Rick



Abbildung 2: OLED-Display

Astley's "Never gonna give you up", auch bekannt als "Rick Roll", unten in einer permanenten Schleife angezeigt, um ein Radio zu imitieren.

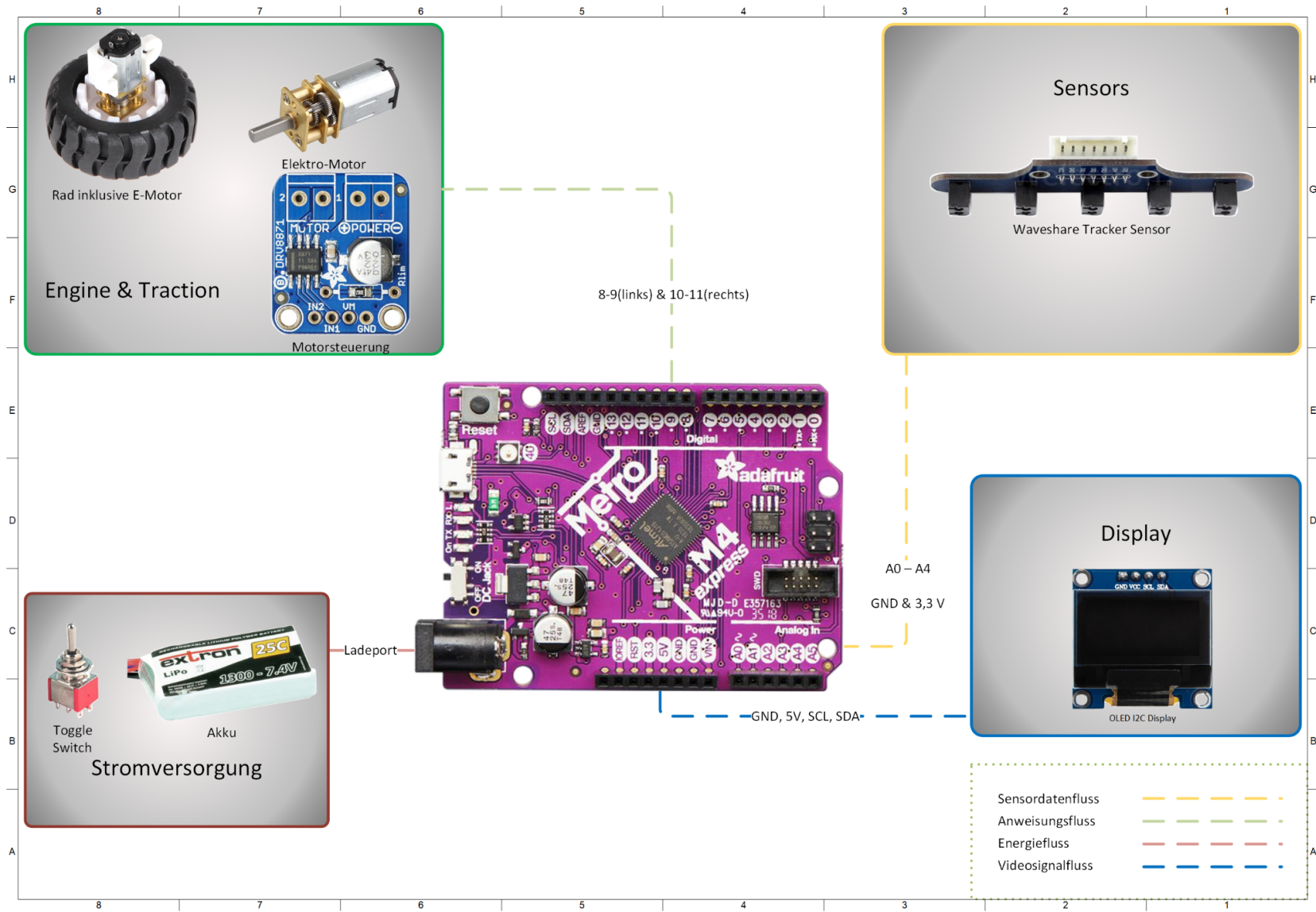
# Zeitplanung

ID	Task Name	Start	Finish	Duration	4. Jul 2021	11. Jul 2021	18. Jul 2021	25. Jul 2021	1. Aug 2021	8. Aug 2021	15. Aug 2021	22. Aug 2021	29. Aug 2021	5. Sep 2021	12. Sep 2021	19. Sep 2021	26. Sep 2021
1	Kickoffmeeting	06/07/2021	06/07/2021	1d	■												
2	Komponentenauswahl	07/07/2021	19/07/2021	13d	■	■	■	■	■	■	■	■	■	■	■	■	■
3	Followup Meeting mit Projektbetreuer	20/07/2021	20/07/2021	1d			■										
4	Abholen der Komponenten	22/07/2021	22/07/2021	1d				■									
5	Testen des Roboters	23/07/2021	23/07/2021	1d				■									
6	Erweitern des Roboters mit Zusatzkomponenten	24/07/2021	02/08/2021	10d				■	■	■	■	■	■	■	■	■	■
7	Programmieren des Roboters	24/07/2021	07/09/2021	45d				■	■	■	■	■	■	■	■	■	■
8	Testen & Optimieren des Programmcodes	08/09/2021	17/09/2021	10d										■	■	■	■
9	Erstellen der Dokumentation	18/09/2021	22/09/2021	5d												■	■
10	Abgabe	23/09/2021	23/09/2021	1d													■

Milestone 1:  
Erreichen der  
Funktionalität

Milestone 2:  
Abgabebereit

# Schaltplan / Verkabelungsplan / Anschlussplan



# Beschreibung aller verbauten Teile

## **Adafruit Metro M4**

Der Adafruit Metro M4 ist das Board des Roboters. Er besitzt den Microchip ATSAMD51 und dazu folgende Bestandteile:

Cortex M4-Kern mit 120 MHz

512 KB Flash, 192 KB RAM

32-Bit, 3,3 V Logik und Stromversorgung

Analoge und digitale Pins

PWM-Ausgänge

Stereo-I2S-Eingang/Ausgang mit MCK-Pin

10-Bit-Parallel-Capture-Controller (für Kamera/Video-Eingang)

Eingebaute Krypto-Engines mit AES (256 Bit), echtem RNG, Pubkey-Controller

## **Gedruckte Halterung für Metro M4**

Eine Halterung für den Adafruit Metro M4, die mit einem 3D Drucker hergestellt wurde

## **Chassis-Komponenten**

Die Chassis des Roboters wurde mit einem Laser hergestellt.

## **Sicherungshalter mit Sicherung**

Die Sicherung schützt den Stromkreis vor Überlastung und verhindert damit Schäden

## **Kippschalter**

Über den Kippschalter am hinteren Teil des Roboters wird das Fahrzeug eingeschaltet

## **Akku**

Für die Stromversorgung des Roboters wurde folgender Akku eingebaut:

LiPo, 7.4V, 1300, Extron, XT-60

### **Mini Metallgetriebe-Motor inkl. Rad und Halter**

Die Motoren mit Rad und Halterung sind von Joy-IT und können einzeln über das Adafruit Metro M4 angesprochen werden

### **Adafruit DRV8871**

Der Adafruit DRV8871 ist der Motor Driver, über den die Motoren des Roboters gesteuert werden können.

### **Waveshare Tracker Sensor**

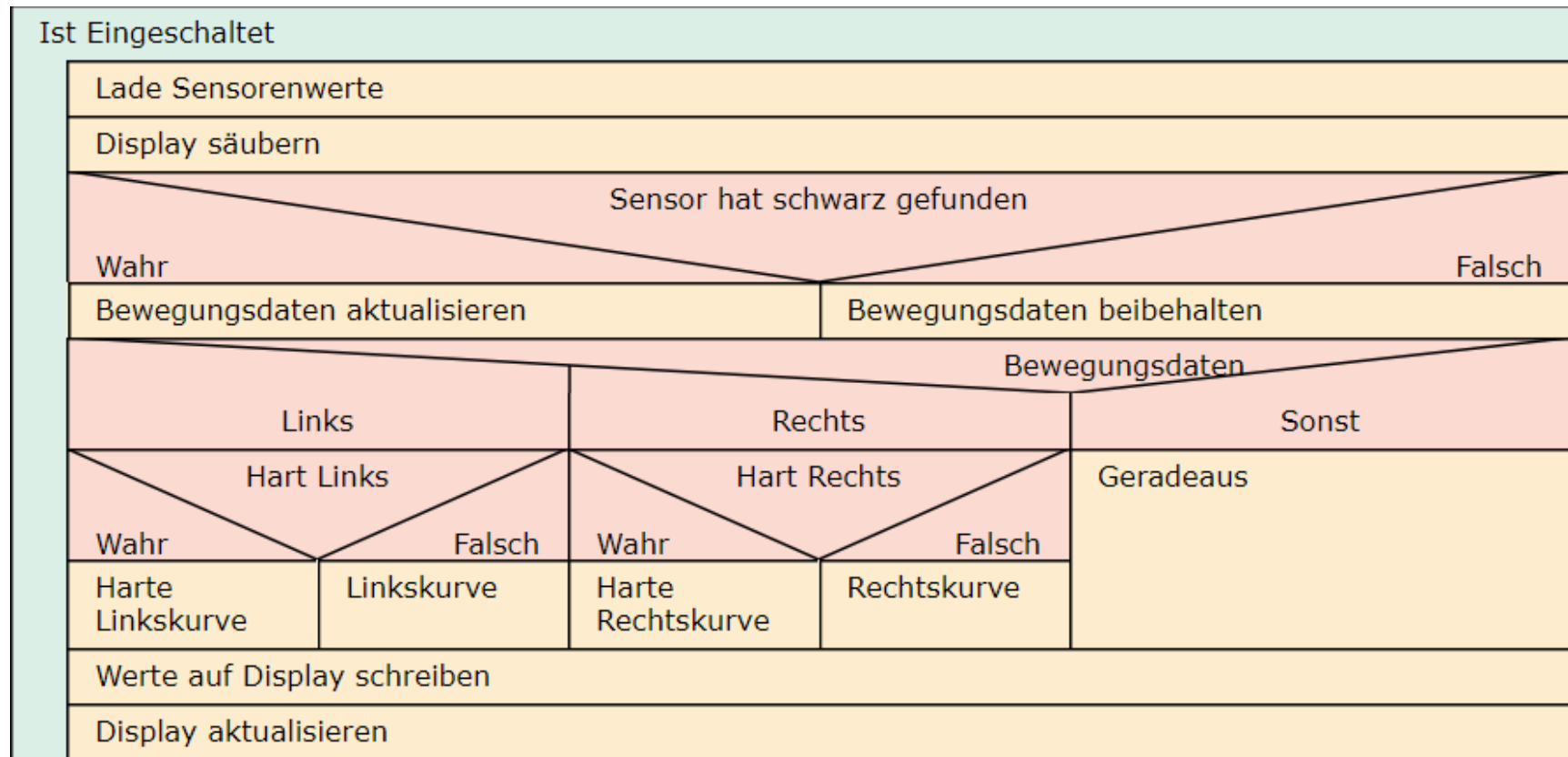
Der Tracker besitzt 5 Sensoren, die über die Analogen Pins A0-A4 mit dem Metro M4 verbunden werden. Über die Sensoren wird die Richtung bestimmt, in die der Roboter fahren muss.

### **OLED 0.96" Display**

Bei dem verbauten Display handelt es sich um einen AZDelivery 0,96 Zoll OLED Display mit einem I2C SSD1306 Chip und einer Auflösung von 128 x 64 Pixeln







## Struktogramm



## Detaillierte Beschreibung des Codes

Der Code für die Bedienung des Roboters ist in 4 Dateien aufgeteilt. Dabei ist die code.py Datei die Main-Datei, die beim Start des Roboters ausgeführt wird und die auf die Funktionen der anderen Dateien zurückgreift. Die Datei Motor.py liefert die Funktionen für das ansteuern des Motorcontrollers, sensoren.py führt die Abfrage der Sensoren zu und oled.py liefert Helper-Funktionen für die Anzeige von Werten auf dem Display.

 code.py motor.py oled.py sensoren.py

### code.py:

```
# Ensures that the amount passed in is within the valid boundries of the speed.
def get_valid_speed(amount):
    if amount > 0 and amount < 100:
        return int(amount)
    elif amount < 0:
        return 0
    elif amount > 100:
        return 100

# Define the max speed of the robot. (0 - 100)
max_speed = 25

# Define the hard turn multilplier when driving a sharp turn.
hard_turn_multiplier = 1.25

# Define the slow turn multilplier when driving a soft turn.
slow_turn_multiplier = .5

# Calculate sharp turn speed.
max_turn_speed = get_valid_speed(max_speed * hard_turn_multiplier)
slow_turn_speed = get_valid_speed(slow_turn_multiplier * max_speed)
```

Nach dem Import der Funktionen aus den anderen Python Dateien werden hier die Geschwindigkeiten für die Kurven mittels festgelegten Multiplikatoren berechnet. Durch die Multiplier kann mittels Anpassung einer Variable die Geschwindigkeit aller Fahrweisen

angepasst werden. Damit die Werte nicht über 100 oder unter 0 liegen und es sich um einen Integer handeln muss, werden diese Anforderungen in einer Helper-Funktion sichergestellt..

```
# Default instructions for movement.
forward = True
left = False
hard_left = False
right = False
hard_right = False

while True:

    print("begin loop: ", int(hard_left), left, forward, right, hard_right)

    # Read the sensor Values.
    sensorWerte = sensorAbfrage()
    sensorWert_RR = sensorWerte[4]
    sensorWert_R = sensorWerte[3]
    sensorWert_M = sensorWerte[2]
    sensorWert_L = sensorWerte[1]
    sensorWert_LL = sensorWerte[0]

    print('sensorwert: ', sensorWerte)

    # Update the movement instructions if any sensor detects the black line
    # If the black line is detected nowhere the robot continues it's current instructions.
    if 1 in sensorWerte:

        print('updated movement instructions')
        forward = bool(sensorWert_M)

        left = bool(sensorWert_L or sensorWert_LL)
        hard_left = bool(sensorWert_LL)

        right = bool(sensorWert_R or sensorWert_RR)
        hard_right = bool(sensorWert_RR)

    clear_display()
```

Die Variablen für die Steuerung werden in dem Scope außerhalb der While Schleife definiert. Somit können diese Werte zwischen wiederholungen des loops beibehalten werden. Anschließend werden die Variablen für die Richtungssteuerung angepasst. Die Richtungssteuerung Variablen werden jedoch nur angepasst, wenn mindestens einer der Sensoren eine Schwarze Oberfläche erkennt.

```

if left:

    motor_left_speed = 0 if hard_left or not forward else slow_turn_speed
    motor_right_speed = max_turn_speed if hard_left else max_speed

    # Drive turn with max_turn speed if a sharp turn is required
    motorR(motor_right_speed)

    # Stop the left motor if a sharp turn is required.
    # Otherwise only slow the motor to the in the slow turn multiplier defined speed
    motorL(motor_left_speed)

    print("drive left", ('l:', motor_left_speed), ('r:', motor_right_speed))
    print_motor_speed(speed_r=motor_right_speed, speed_l=motor_left_speed)
    print_movement_instruction('left' if not hard_left else 'hard left')

elif right:

    motor_left_speed = max_turn_speed if hard_right else max_speed
    motor_right_speed = 0 if hard_right or not forward else slow_turn_speed

    # Drive turn with max_turn speed if a sharp turn is required
    motorR(motor_right_speed)

    # Stop the right motor if a sharp turn is required.
    # Otherwise only slow the motor to the in the slow turn multiplier defined speed
    motorL(motor_left_speed)

    print("drive right", ('l:', motor_left_speed), ('r:', motor_right_speed))
    print_motor_speed(speed_r=motor_right_speed, speed_l=motor_left_speed)
    print_movement_instruction('right' if not hard_right else 'hard right')

else:

    motorR(max_speed)
    motorL(max_speed)
    print("drive forward", ('l:', max_speed), ('r:', max_speed) )
    print_motor_speed(speed_r=max_speed, speed_l=max_speed)
    print_movement_instruction('forward')

update_display()
print('end loop: ', hard_left, left, forward, right, hard_right)

```

Hier werden nun basierend auf den Variablen die Motoren gesteuert. Dabei werden die Geschwindigkeiten je nach Variable an die motor.py Funktionen weitergegeben. Gleichzeitig

werden die Vorgänge auch in der Konsole und auf dem OLED Display angezeigt. Die Funktionen des Motors steuern lediglich den rechten und linken Motor, weshalb hier die motor.py aus dem Standardcode übernommen werden konnte. In der sensor.py wurde lediglich der Grenzwert zu Testzwecken auf 200 reduziert, da der Untergrund der genutzten Teststrecke nicht weiß, sondern hellgrau ist.

```
def is_white(analogWert):  
    grenzwert_Weiss = 200
```

### **oled.py**

```
import board  
import adafruit_ssd1306  
import busio as io  
import time  
  
width = 128  
height = 64  
  
i2c = io.I2C(board.SCL, board.SDA)  
oled = adafruit_ssd1306.SSD1306_I2C(width, height, i2c, addr=0x3c)
```

Für die Nutzung des Displays wurde die library “adafruit\_ssd1306”, sowie deren dependency “adafruit\_circuitpython\_framebuf”, auf dem Roboter installiert. Damit kann das OLED Display wie im obigen Screenshot zu erkennen als Variable instanziiert und später angesprochen werden.

```

def update_display():
    update_rick()
    oled.show()

def clear_display():
    oled.fill(0)

def print_movement_instruction(direction):
    oled.text(direction, 0, 0, 1)

def print_motor_speed(speed_r, speed_l):
    oled.text('Speed L: %s' % (speed_l), 0, 10, 1)
    oled.text('Speed R: %s' % (speed_r), 0, 20, 1)

def update_rick():
    global last_updated_rick_at
    global word

    oled.text("Playing: 'Rick Roll'", 0,40,1)
    oled.text(" ".join(rick_roll[word % len(rick_roll):word+3 % len(rick_roll)]), 0, 40, 1)

    if int(time.time()) - last_updated_rick_at >= 2:
        last_updated_rick_at = int(time.time())
        word += 3

```

Außerdem wurden einige Helper-Funktionen für das Anzeigen von Werten auf dem display definiert. Diese können verwendet werden um die Entsprechenden Werte auf dem Display anzuzeigen. Es wird die Richtung, die Geschwindigkeit der jeweiligen Motoren sowie ein Autoradio angezeigt. Das Autoradio gibt einen Liedtext in 3-Wort schritten wieder.

## Poster

