HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Fakultät Ingenieurwissenschaften

Echtzeitsysteme und Mobile Robtik

Echtzeitsysteme und mobile Robotik

Author Nico Beyer

15. Mai 2025

Inhaltsverzeichnis

1 Einleitung

Im Rahmen dieses Praktikums sollen grundlegende Kenntnisse im Umgang mit ROS 2 (Robot Operating System) vermittelt und anhand praxisnaher Beispiele angewendet werden. Das Projekt gliedert sich dabei in mehrere aufeinander aufbauende Schritte:

1. **Einführung in ROS 2**
   Zunächst wird ROS 2 installiert und die grundlegende Kommunikation anhand der bekannten `turtlesim`-Simulation erlernt. Ziel ist es, erste Publisher- und Subscriber-Knoten zu erstellen und die Prinzipien des ROS-Netzwerks zu verstehen.

2. **Erstellung eines eigenen ROS 2-Pakets**
   Aufbauend auf dem Wissen aus der Simulation wird ein eigenes ROS 2-Paket erstellt. Innerhalb dieses Pakets soll ein kleines Programm geschrieben werden, mit dem die `turtlesim` aktiv gesteuert wird. Dadurch werden grundlegende ROS-Konzepte wie Topics, Nodes und Launchfiles vertieft.

3. **Programmierung eines eigenen Roboters**
   Anschließend wird ein echter, mobiler Roboter mit differenziellem Antrieb programmiert. Die benötigte Hardware wird dabei fertig zur Verfügung gestellt. Die Teilnehmenden programmieren einen ESP32-Mikrocontroller, welcher mit verschiedensten Sensoren und Aktoren verbunden ist. Dieser hat zusätzlich die Möglichkeit, per WLAN ins ROS 2-Netzwerk eingebunden zu werden. Im Fokus steht die Umsetzung der Fahrfunktionalität und die Einbindung ins ROS Netzwerk.

4. **Erweiterung durch Sensorik und eigene Projekte**
   Sobald die Basisfunktionen des Roboters implementiert sind, können die verbauten Sensoren (z.B. Ultraschall, Infrarot oder IMU) verwendet werden, um eigene kleine Projekte umzusetzen. Ziel ist es, kreative Ideen zu entwickeln und diese selbstständig im ROS-Kontext zu realisieren.

5. **Arbeiten mit einem professionellen Roboter: TurtleBot 4**
   Zum Abschluss wird der professionelle Roboter *TurtleBot 4* eingesetzt. Dabei sollen die bisher erlernten Kenntnisse angewendet und vertieft werden. Der TurtleBot bringt eine vorkonfigurierte ROS 2-Umgebung mit, inklusive Lokalisierung und Navigation. Diese Funktionen werden getestet und für kleine, eigenständige Projekte genutzt – idealerweise angelehnt an die zuvor entwickelten Anwendungen auf dem eigenen Roboter.

Der gesamte Projektverlauf wird in Form eines schriftlichen Belegs dokumentiert. Dabei sollen sowohl **technische Details als auch eigene Erkenntnisse und Reflexionen** festgehalten werden.

2 Durchführung

2.1 ROS2 Grundlagen

Die **Installation** muss auf einem Ubuntu-22.04-System erfolgen. Falls Sie einen Windows-Rechner verwenden, können Sie eine virtuelle Maschine erstellen. Eine Anleitung dazu finden Sie in Abschnitt 4.1.1.

Verfügen Sie bereits über ein Ubuntu-22.04-System, können Sie direkt mit der Installation von ROS 2 beginnen (siehe Abschnitt 4.1.2).

Im Anschluss sollten Sie sich eigenständig mit den **Grundlagen von ROS 2** vertraut machen. Die offizielle ROS-Dokumentation bietet dafür eine ausgezeichnete Einführung: LINK [11]. Zusätzlich stehen im Internet zahlreiche Tutorials und Lernressourcen zur Verfügung, die bei der Einarbeitung unterstützen können.

Ein besonderer Fokus sollte auf der **turtlesim**-Simulation liegen, da viele grundlegende Konzepte von ROS 2 – etwa Kommunikation über Topics und das Zusammenspiel von Nodes – dort anschaulich vermittelt werden: LINK [5].

Darüber hinaus empfiehlt es sich, gezielt die Funktionsweise von **Nodes** (LINK [17]) und **Topics** (LINK [18]) zu vertiefen – diese Konzepte bilden das Fundament für alle weiteren Aufgaben im Praktikum.

2.2 Erstellung eines eigenen ROS 2-Pakets

In diesem Abschnitt erstellen Sie Ihr erstes **eigenes ROS 2-Paket**.

Arbeiten Sie dazu zunächst die folgende Dokumentation durch: LINK [1]

Besonders wichtig sind dabei die folgenden Themen:

- Erstellen eines Arbeitsbereichs: LINK [2]
- Erstellen eines eigenen ROS 2-Pakets: LINK [3]
- Erstellen einer Publisher Subscriber Kommunikation (Falls Sie C++ bevorzugen): LINK [20]
- Erstellen einer Publisher Subscriber Kommunikation (Falls Sie Python bevorzugen): LINK [21]

Erstellen Sie nun ein neues Paket oder verwenden Sie das im Tutorial erstellte Paket als Grundlage.

Ihre Aufgabe ist es, eine Node zu entwickeln, die Bewegungsbefehle an das Topic `/cmd_vel` sendet, um damit die Turtle in der `turtlesim`-Simulation zu steuern. Lassen Sie die Turtle beispielsweise ein Dreieck oder ein Viereck fahren.

Hinweis: Dieses Paket wird im weiteren Verlauf des Praktikums erneut verwendet.

2.3 Programmierung eines eigenen Roboters

In diesem Abschnitt programmieren Sie einen eigenen Roboter.

Der Roboter ist mit einem ESP32 ausgestattet, der dazu genutzt wird, die Sensoren und Aktoren des Roboters auszulesen bzw. anzusteuern. Der Grund für den Einsatz eines ESP32 liegt darin, dass dieser direkt mit dem ROS-Netzwerk kommunizieren kann. Dies wird mithilfe der Bibliothek *Micro-ROS* ermöglicht (LINK [6]).

Die **Funktionsweise** sieht folgendermaßen aus: Der ESP32 fungiert als Client und kommuniziert mit einem sogenannten Agenten, der als Server agiert. Der Agent wird auf dem

Host-PC ausgeführt. Sobald eine Verbindung zwischen ESP32 und Agent hergestellt ist, überträgt der ESP32 automatisch seine Informationen an den Agenten. Dieser übersetzt die empfangenen Nachrichten direkt ins ROS-Netzwerk.

Um den Agenten zu installieren, folgen Sie bitte der Anleitung in Abschnitt 4.1.3, sofern Sie dies noch nicht erledigt haben.

Die Programmierung des ESP32 erfolgt über *Visual Studio Code* mit der Erweiterung *PlatformIO*. Es gibt zwar weitere Möglichkeiten zur Programmierung, in diesem Praktikum wird jedoch ausschließlich dieser Weg verwendet. Eine detaillierte Anleitung zur Programmierung des ESP32 finden Sie in Abschnitt 4.1.4.

Aufgabe: Nutzen Sie das dort gegebene Beispielprogramm, um den Roboter so zu programmieren, dass er auf das Topic `/cmd_vel` hört. Damit können Sie den Roboter zunächst manuell steuern.

Anschließend sollen Sie das in Abschnitt 2.2 erstellte Paket verwenden, mit dem in der `turtlesim`-Simulation eine kleine Schildkröte automatisch eine bestimmte Form (z.B. ein Dreieck oder Viereck) abgefahren hat. Führen Sie die Node genau wie zuvor aus, jedoch ohne die `turtlesim`-Simulation zu starten. Beobachten Sie das Verhalten des Roboters und halten Sie Ihre Beobachtungen schriftlich fest.

2.4 Erweiterung durch Sensorik und eigene Projekte

Nachdem die Grundfunktionalität der Fortbewegung Ihres Roboters funktioniert, können Sie nun zusätzliche Funktionen implementieren. Hierfür können Sie die auf dem Roboter verbauten Sensoren nutzen. Eine Übersicht der verfügbaren Sensoren finden Sie in Abschnitt 4.2.1.

Aufgabe: Wählen Sie mindestens einen der aufgeführten Sensoren aus und entwickeln Sie damit ein kleines Projekt. Der ESP32 soll dabei die Sensorwerte erfassen und diese in das ROS-Netzwerk publizieren. Die Auswertung und Weiterverarbeitung der Sensordaten soll auf dem Host-PC erfolgen.

Dokumentieren Sie Ihre Umsetzung und beschreiben Sie kurz den Aufbau, die Funktionsweise und mögliche Herausforderungen.

2.5 Arbeiten mit einem professionellen Roboter

Nachdem Sie nun die Grundlagen von ROS 2 kennengelernt und erfolgreich einen eigenen Roboter programmiert haben, arbeiten Sie in diesem Abschnitt mit einem professionellen Roboter: dem **TurtleBot 4** (LINK [12]).

Die Funktionsweise des TurtleBot 4 ähnelt stark der Ihres selbstgebauten Roboters. Sobald der TurtleBot gestartet ist, verbindet er sich mit dem WLAN und erzeugt automatisch mehrere Nodes, die unterschiedliche Topics öffnen. Diese Topics dienen zum Publizieren von Sensordaten sowie zum Empfangen von Steuerbefehlen.

Somit können Sie mit dem TurtleBot auf sehr ähnliche Weise arbeiten wie mit Ihrem eigenen Roboter.

Aufgaben für Zuhause:

1. Verschaffen Sie sich einen Überblick über die Funktionsweise des TurtleBot 4, indem Sie das offizielle Benutzerhandbuch durchgehen: LINK [14]
2. Nutzen Sie die Simulationsumgebung, um den TurtleBot bereits virtuell zu steuern: LINK [13]
3. Machen Sie sich mit den grundlegenden Funktionen vertraut:
   - Bewegung des Roboters über das Topic `/cmd_vel`
   - Visualisierung der Sensordaten in `RViz`

Aufgaben vor Ort:

1. Starten Sie den realen TurtleBot 4 und steuern Sie ihn zunächst manuell, genau wie in der Simulation.
2. Nutzen Sie `RViz`, um Sensordaten zu visualisieren, und die Kommandozeile, um Informationen aus Topics auszulesen.
3. Führen Sie eine Kartierung der Umgebung mit Hilfe des TurtleBot durch. Folgen Sie dazu der Anleitung: LINK [4]

   Speichern Sie die erstellte Karte für den nächsten Schritt.
4. Navigieren Sie den TurtleBot anschließend autonom auf der aufgenommenen Karte. Anleitung: LINK [9]

   Beobachten und bewerten Sie dabei das Verhalten des Roboters während der autonomen Navigation.
5. Überlegen Sie sich nun eine kleine Beispielanwendung, die der TurtleBot ausführen soll. Wenn möglich, verwenden Sie das von Ihnen zuvor entwickelte Programm für den kleinen Roboter als Grundlage.
6. Optional: Erweitern Sie Ihr bestehendes Programm, um es auf die Fähigkeiten des TurtleBot 4 anzupassen.

3 Auswertung

Am Ende des Praktikums sollen die durchgeführten Arbeiten in einem **schriftlichen Beleg** dokumentiert werden. Dieser dient dazu, die erlernten Inhalte, die umgesetzten Methoden sowie die erzielten Ergebnisse systematisch aufzuarbeiten und kritisch zu reflektieren.

Ein besonderer Fokus liegt dabei auf den **selbstständig entwickelten Anwendungen** im Rahmen der Sensorintegration und Projektideen. Diese Projekte zeigen nicht nur den praktischen Umgang mit ROS 2 und den Sensoren des eigenen Roboters, sondern spiegeln auch die Kreativität und das Verständnis der Teilnehmenden für robotische Systeme wider. Neben einer technischen Beschreibung der Umsetzung sollen dabei auch Herausforderungen, Lösungsansätze und ggf. Verbesserungsvorschläge thematisiert werden.

Zusätzlich zur schriftlichen Dokumentation soll der im Praktikum entwickelte **Quellcode vollständig abgegeben** werden. Hierzu ist ein öffentlich zugängliches GitHub-Repository

anzulegen, in dem sämtliche relevanten Dateien, Konfigurationen und Dokumentationen enthalten sind. Der Link zu diesem Repository muss im Beleg eindeutig angegeben sein.

Die Abgabe des GitHub-Repositories sowie des schriftlichen Belegs bildet den Abschluss des Praktikums.

4 Hilfestellung und Dokumentation

4.1 Einrichtung der Arbeitsumgebung

4.1.1 Einrichtung einer virtuellen Machine

In dieser Anleitung beschreiben wir Schritt für Schritt, wie Sie unter Windows mit VirtualBox eine virtuelle Maschine erstellen und Ubuntu 22.04 LTS als Gastbetriebssystem installieren.

1. **VirtualBox herunterladen und installieren**

   - Besuchen Sie die offizielle Seite von VirtualBox: LINK [19]
   - Wählen Sie das passende Installationspaket für Ihr Host-Betriebssystem (Windows, macOS, Linux).
   - Starten Sie das Installationsprogramm und folgen Sie den Anweisungen auf dem Bildschirm.

2. **Ubuntu 22.04 ISO herunterladen**

   - Rufen Sie die Ubuntu-Downloadseite auf: LINK [16]
   - Laden Sie die Desktop-ISO herunter.

3. **Neue virtuelle Maschine anlegen**

   (a) Öffnen Sie VirtualBox und klicken Sie auf `Neu`.
   (b) Vergeben Sie einen **Namen** (z. B. „Ubuntu 22.04“).
   (c) Bei ISO Image wählen die eben heruntergeladene Datei aus.
   (d) Klicken Sie auf `Weiter`.

4. **Arbeitsspeicher (RAM) und Festplatte konfigurieren**

   - Wählen Sie eine geeignete RAM-Größe (empfohlen: mindestens 2048 MB).
   - Legen Sie eine virtuelle Festplatte an (empfohlen: mindestens 20 GB, VDI, dynamisch alloziert).
   - Bestätigen Sie Ihre Auswahl mit `Erzeugen`.

5. **Anpassungen vor dem Start**

   - **Gemeinsame Zwischenablage**: Reiter `Allgemein` → `Erweitert` → „Gemeinsame Zwischenablage“ auf `Bidirektional` setzen.
   - **Netzwerk**:
     - Adapter 1: `NAT`
     - Adapter 2: `Netzwerkbrücke` (optional, für direkten Zugriff ins LAN)
   - **Gemeinsame Ordner**: Reiter `Gemeinsame Ordner` → `Hinzufügen` → wählen Sie einen Ordner auf dem Host, aktivieren Sie „Automatisch bereitstellen“.

6. **Virtuelle Maschine starten**

4.1.2 Installation von ROS 2 Humble

Die Installation von ROS 2 Humble erfolgt gemäß der offiziellen Anleitung unter: LINK [10]
.

Bitte folgen Sie dabei den angegebenen Schritten sorgfältig. Nach Abschluss der grundlegenden Einrichtung wird empfohlen, die vollständige Desktop-Version von ROS 2 zu installieren. Dadurch werden neben den Basisfunktionen auch nützliche Werkzeuge wie `RViz` und `Gazebo` direkt mit installiert.

Nach der Installation sollte zusätzlich das Paket `ros-dev-tools` installiert werden, da es unter anderem das Build-Tool `colcon` enthält, welches im weiteren Verlauf des Praktikums benötigt wird.

Test der Installation Um sicherzustellen, dass ROS 2 korrekt installiert wurde, sollen die mitgelieferten Beispielknoten getestet werden. Folgen sie dazu der Anleitung am Ende der Installationsseite. In einem Terminal wird ein `talker`-Knoten ausgeführt, der Nachrichten verschickt, im anderen ein `listener`-Knoten, der diese Nachrichten empfängt.

Erscheinen im zweiten Terminal empfangene Nachrichten, zeigt dies, dass die ROS-Kommunikation erfolgreich funktioniert und die Installation korrekt abgeschlossen wurde. ROS 2 ist damit bereit für die weiteren Aufgaben im Praktikum.

4.1.3 Installation: Micro-ROS-Agent

Für die Kommunikation zwischen dem ESP32 und dem ROS 2-Netzwerk wird der Micro-ROS-Agent benötigt. Die Installation erfolgt in mehreren Schritten gemäß der offiziellen Anleitung unter LINK [8] .

Zunächst muss die ROS-Umgebung geladen und ein entsprechender Workspace vorbereitet werden:

```
source /opt/ros/humble/setup.bash
mkdir -p micro_ros_ws/src
cd micro_ros_ws/src
git clone -b humble https://github.com/micro-ROS/micro_ros_setup.git
cd ..
```

Hinweis: Sollte Git noch nicht installiert sein, kann dies mit folgendem Befehl nachgeholt werden:

```
sudo apt install git
```

Anschließend müssen weitere Abhängigkeiten installiert und initialisiert werden:

```
sudo apt install python3-rosdep2
sudo apt update && rosdep update
rosdep install --from-paths src --ignore-src -y
```

Nun wird der Workspace mit `colcon` gebaut:

```
colcon build
source install/local_setup.bash
```

Sobald der Workspace erfolgreich gebaut wurde, wird das Agent-Paket erzeugt und kompiliert:

```
ros2 run micro_ros_setup create_agent_ws.sh
ros2 run micro_ros_setup build_agent.sh
source install/local_setup.sh
```

4.1.4 Programmierung des ESP32

Nachdem der Micro-ROS-Agent erfolgreich installiert wurde, muss nun der ESP32 so programmiert werden, dass er sich mit diesem Agenten verbinden kann. Dafür wird das Repository unter LINK [7] verwendet.

Zur Entwicklung wird die Nutzung von **Visual Studio Code** in Kombination mit **PlatformIO** empfohlen.

Einrichtung von Visual Studio Code und PlatformIO

1. Installieren Sie Visual Studio Code über die offizielle Website: LINK
2. Starte Visual Studio Code und öffne den Reiter *Extensions* (Erweiterungen) auf der linken Seite.
3. Suche nach `PlatformIO` und installiere die Erweiterung.
4. Nach der erfolgreichen Initialisierung ist die Entwicklungsumgebung einsatzbereit.

Erstellen eines ersten Projekts

1. Klicke in Visual Studio Code auf das PlatformIO-Symbol (Alienkopf) in der linken Leiste.
2. Öffne das *PlatformIO Home*-Menü und wähle *Open*.
3. Erstelle ein neues Projekt über den Button *Projekt erstellen*.
4. Vergib einen Projektnamen und wähle als Board `Espressif ESP32 Dev Module`.
5. Das Framework bleibt auf `Arduino` eingestellt.
6. Klicke auf `Finish`. Das Projekt wird nun erstellt – dies kann einige Minuten in Anspruch nehmen.

Micro-ROS auf dem ESP32 ausführen

1. Stelle sicher, dass alle benötigten Pakete installiert sind:

   ```
   sudo apt install -y git cmake python3-pip
   ```

2. Falls noch nicht geschehen, erstelle ein neues PlatformIO-Projekt wie im vorherigen Abschnitt beschrieben.

3. Im Projekt sind nun zwei zentrale Dateien wichtig:

   - `platformio.ini` – Konfigurationsdatei für das Projekt
   - `main.cpp` im `src`-Verzeichnis – enthält den Hauptprogrammcode

4. In der `platformio.ini` müssen die Micro-ROS-Bibliotheken eingebunden werden. Füge hierzu folgende Zeile ein:

   ```
   lib_deps =
       https://github.com/micro-ROS/micro_ros_platformio
   ```

5. Weitere Bibliotheken können auf dieselbe Weise eingebunden werden.

Erste Tests mit Micro-ROS auf dem ESP32

Für einen ersten Funktionstest kann folgender Beispielcode verwendet werden. Damit wird eine einfache Verbindung über USB hergestellt, und der ESP32 publiziert regelmäßig Integer-Werte auf ein Topic.

Der Beispielcode initialisiert eine ROS 2-Node auf dem ESP32, erstellt einen Publisher für Ganzzahlen (`Int32`) und sendet jede Sekunde eine Nachricht, deren Wert bei jedem Durchlauf um eins erhöht wird. Die Kommunikation erfolgt dabei über eine serielle Verbindung.

Der vollständige Quelltext:

```
// Standard Arduino-Funktionen
#include <Arduino.h>
// Micro-ROS PlatformIO-Anbindung
#include <micro_ros_platformio.h>

// ROS 2 Kernfunktionen
#include <rcl/rcl.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>

// Standard ROS 2 Nachrichtenformat (Int32)
#include <std_msgs/msg/int32.h>

// Sicherstellen, dass Arduino Serial Transport verwendet wird
#if !defined(MICRO_ROS_TRANSPORT_ARDUINO_SERIAL)
```

```
#error This example is only available for Arduino framework with serial transport.
#endif

// Definition der ROS-Objekte
rcl_publisher_t publisher;
std_msgs__msg__Int32 msg;

rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

// Makros zur Fehlerbehandlung
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

// Fehlerbehandlungsfunktion
void error_loop() {
  while(1) {
    delay(100);
  }
}

// Timer-Callback-Funktion
void timer_callback(rcl_timer_t * timer, int64_t last_call_time) {
  RCLC_UNUSED(last_call_time);
  if (timer != NULL) {
    RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
    msg.data++;
  }
}

void setup() {
  Serial.begin(115200);
  set_microros_serial_transports(Serial);
  delay(2000);

  allocator = rcl_get_default_allocator();

  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
  RCCHECK(rclc_node_init_default(&node, "micro_ros_platformio_node", "", &support));

  RCCHECK(rclc_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "micro_ros_platformio_node_publisher"));

  const unsigned int timer_timeout = 1000;
  RCCHECK(rclc_timer_init_default(
```

```
    &timer,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback));

  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_timer(&executor, &timer));

  msg.data = 0;
}

void loop() {
  delay(100);
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

Anschließend kann die Verbindung zwischen dem ESP32 und dem Micro-ROS-Agent hergestellt werden:

- **Verbindung über USB:** Zuerst ein neues Terminal öffnen und in das Micro-ROS-Workspace-Verzeichnis (`micro_ros_ws`) wechseln. Dort muss die lokale `setup`-Datei gesourced werden:

  ```
  source install/local_setup.bash
  ```

  Danach kann der Micro-ROS-Agent im Serial-Modus gestartet werden:

  ```
  ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0
  ```

  Dadurch wird die Kommunikation über die serielle Schnittstelle ermöglicht.

- **Verbindung über WiFi:** Alternativ kann eine Verbindung über das Netzwerk (UDP) hergestellt werden. Auch hier gilt: zuerst das Terminal öffnen, in den Micro-ROS-Workspace wechseln und die lokale `setup`-Datei sourcen:

  ```
  source install/local_setup.bash
  ```

  Anschließend kann der Micro-ROS-Agent im UDP-Modus gestartet werden:

  ```
  ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888
  ```

  Für die Verbindung über WiFi müssen jedoch zuvor Anpassungen im ESP32-Programmcode vorgenommen werden, um den Transport auf WiFi umzustellen.

4.2 Dokumentation der Hardware

4.2.1 Übersicht der verbauten Komponenten

Komponente	Funktion / Beschreibung
ESP32	Zentrale Steuereinheit des Roboters, verbindet sich über WLAN mit dem ROS2-Netzwerk.
GY-9250 (IMU)	Misst Beschleunigung, Rotation und Magnetfeld – nützlich für Lagebestimmung.
HC-SR04 (Ultraschall)	Misst Distanzen im Frontbereich mittels Ultraschallimpulsen.
APDS-9960	Farb-, Gesten- und Näherungssensor – kann für Linienfolgen oder Farberkennung genutzt werden.
Mikroschalter (Bumper)	Dient als taktiler Sensor zur Kollisionserkennung oder als Endschalter.
IR-Kommunikationsmodul	Ermöglicht einfache drahtlose Kommunikation zwischen Robotern über Infrarot.
18650 Akku + Lademodul	Versorgt den Roboter mit Energie. Unterstützt Ladeströme bis 1A, Versorgung bis 2A.
Rasperry Pi Header	Möglichkeit einen Rasperry Pi anzuschließen
Passiver Piezo Buzzer	Ermöglicht einfache akustische Signale wie Warn- oder Hinweistöne.
OLED Display Modul 128×64 Pixel	I2C Display zur Anzeige von Statusinformationen, Sensorwerten oder Debug-Ausgaben.
LED	Signalausgabe über Licht – z. B. zur Statusanzeige oder für einfache Benutzerinteraktion.

Tabelle 1: Übersicht der verbauten Hardwarekomponenten auf dem Roboter

4.2.2 Motoren

Als Antriebsmotoren kommen in diesem Roboter zwei 28BYJ-48 Schrittmotoren in Kombination mit ULN2003-Treiberplatinen zum Einsatz. Diese Motoren zeichnen sich durch folgende Eigenschaften aus:

- Da es sich um Schrittmotoren handelt, ist eine präzise Steuerung der Bewegung ohne zusätzliche Sensorik wie Encoder möglich.
- Eine separate Regelung der Motoren ist nicht notwendig, da die Position direkt über die Anzahl der Schritte bestimmt werden kann.

Die Motoren ermöglichen somit eine exakte Ansteuerung ohne komplexe Rückkopplungs- oder Regelsysteme.

Ansteuerung

Die Ansteuerung erfolgt, indem die vier Spulen des Motors über die Pins IN1 bis IN4 der ULN2003-Treiberplatine in einer bestimmten Reihenfolge angesteuert werden. Abbildung

1 zeigt den Aufbau des Motors:

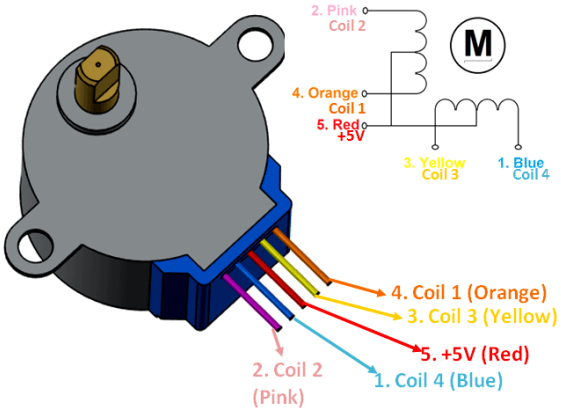

Abbildung 1: Aufbau des 28BYJ-48 Schrittmotors (Quelle: [15])

Die folgende Tabelle zeigt das Ansteuermuster laut Datenblatt:

IN1	IN2	IN3	IN4
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	0	0
1	1	0	0
1	0	0	0
1	0	0	1

Die Generierung dieser Schrittfolgen erfordert ein kontinuierliches und präzises Timing. Würde der Hauptcontroller (z.B. ein ESP32) dies übernehmen, wäre er dauerhaft mit der Erzeugung der Steuersignale beschäftigt und könnte keine weiteren Aufgaben ausführen.

Aus diesem Grund wurde pro Motor ein zusätzlicher Mikrocontroller, ein ATtiny 212, vorgesehen. Dieser übernimmt die Generierung der Schrittimpulse und kann über UART-Befehle vom Hauptcontroller angesteuert werden. Der ESP32 nutzt dafür seinen zweiten Hardware-UART. Da nur gesendet wird, wird lediglich der TX-Pin (GPIO17) verwendet.

Die Kommunikation erfolgt über folgendes Nachrichtenformat:

```
(uint8_t) START_BYTE (0xAA)
(uint8_t) id
(uint8_t) mode
(uint8_t) dir
(uint16_t) param
```

- **id** – Adresse des Zielmotors
- **mode** – Betriebsmodus des Motors
- **dir** – Drehrichtung

- **param** – Modusabhängiger Parameter (16-Bit)

Derzeit sind folgende Betriebsmodi implementiert:

Mode	Beschreibung
0	Schrittmodus – Der Motor führt eine feste Anzahl an Schritten aus
1	Dauerbetrieb – Der Motor dreht kontinuierlich

Im Feld `dir` kann die Drehrichtung (vorwärts/rückwärts) angegeben werden. Der Parameter `param` bestimmt abhängig vom Modus entweder die Schrittanzahl oder die Drehgeschwindigkeit.

4.2.3 Pinbelegung des ESP32

Die folgende Tabelle zeigt die verwendeten Pins des ESP32 sowie deren Zuordnung zu den jeweiligen Sensoren und Komponenten des Roboters:

Komponente	Schnittstelle	ESP32-Pin(s)
IMU (MPU 9265)	I2C	GPIO 21 (SDA), GPIO 22 (SCL) GPIO 34 (INT)
Ultraschallsensor (HC-SR04)	Digital	GPIO 27 (Trigger), GPIO 25 (Echo)
Farb-, Helligkeits- und Abstandssensor (APDS-9960)	I2C	GPIO 21 (SDA), GPIO 22 (SCL)
Bumper	Digital	GPIO 12, GPIO 14
Infrarot-Kommunikation	Digital	GPIO 32 (Sender), GPIO 33 (Empfänger)
Schrittmotor (UART)	UART	GPIO 17 (TX)
PWM	Digital	GPIO 5, GPIO 18, GPIO 19
Passiver Piezo Buzzer	Digital	GPIO 4
OLED Display Modul (128×64 Pixel)	I2C	GPIO 21 (SDA), GPIO 22 (SCL)
LED	Digital	GPIO 15

Tabelle 2: Pinbelegung des ESP32

Bibliography

[1] *Beginner: Client Libraries — ROS 2 Documentation: Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries.html (besucht am 25. Apr. 2025).

[2] *Creating a ROS 2 Workspace — ROS 2 Documentation: Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html (besucht am 25. Apr. 2025).

[3] *Creating Your First ROS 2 Package — ROS 2 Documentation: Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html (besucht am 25. Apr. 2025).

[4] *Generate Map - TurtleBot 4 Tutorial*. URL: https://turtlebot.github.io/turtlebot4-user-manual/tutorials/generate_map.html (besucht am 25. Apr. 2025).

[5] *Introducing Turtlesim — ROS 2 Documentation: Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Introducing-Turtlesim/Introducing-Turtlesim.html (besucht am 25. Apr. 2025).

[6] *micro-ROS: Embedded ROS 2 for Microcontrollers*. URL: https://micro.ros.org (besucht am 25. Apr. 2025).

[7] *micro_ros_platformio GitHub Repository*. URL: https://github.com/micro-ROS/micro_ros_platformio/tree/main (besucht am 25. Apr. 2025).

[8] *micro_ros_setup GitHub Repository*. URL: https://github.com/micro-ROS/micro_ros_setup (besucht am 25. Apr. 2025).

[9] *Navigation - TurtleBot 4 Tutorial*. URL: https://turtlebot.github.io/turtlebot4-user-manual/tutorials/navigation.html (besucht am 25. Apr. 2025).

[10] *ROS 2 Humble Installation*. URL: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html (besucht am 25. Apr. 2025).

[11] *ROS 2 Humble Tutorials Overview*. URL: https://docs.ros.org/en/humble/Tutorials.html (besucht am 15. Mai 2025).

[12] *TurtleBot 4 – Clearpath Robotics*. URL: https://clearpathrobotics.com/turtlebot-4/ (besucht am 25. Apr. 2025).

[13] *TurtleBot 4 Simulator*. URL: https://turtlebot.github.io/turtlebot4-user-manual/software/turtlebot4_simulator.html#turtlebot-4-simulator (besucht am 25. Apr. 2025).

[14] *TurtleBot 4 User Manual*. URL: https://turtlebot.github.io/turtlebot4-user-manual/ (besucht am 25. Apr. 2025).

[15] *Tutorial: 28BYJ-48*. URL: https://elektro.turanis.de/html/prj143/index.html (besucht am 25. Apr. 2025).

[16] *Ubuntu 22.04 LTS Release*. URL: https://releases.ubuntu.com/22.04/ (besucht am 25. Apr. 2025).

[17] *Understanding ROS 2 Nodes — ROS 2 Documentation: Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html (besucht am 25. Apr. 2025).

[18] *Understanding ROS 2 Topics — ROS 2 Documentation: Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (besucht am 25. Apr. 2025).

[19] *VirtualBox Downloads*. URL: https://www.virtualbox.org/wiki/Downloads (besucht am 25. Apr. 2025).

[20] *Writing a Simple Publisher and Subscriber (C++) — ROS 2 Documentation: Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html (besucht am 25. Apr. 2025).

[21] *Writing a Simple Publisher and Subscriber (Python) — ROS 2 Documentation: Humble*. URL: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html (besucht am 25. Apr. 2025).