

- Nutzung von micro-RoS zwischen einem Raspberry Pi 5 und einem ESP32
 - 1. Einführung in micro-RoS
 - Was ist micro-RoS?
 - Warum micro-RoS in der Robotik?
 - Architektur
 - 2. Installation von micro-RoS auf Raspberry Pi 5 und ESP32
 - 2.1 Installation auf dem Raspberry Pi 5
 - 1. ROS2 auf dem Raspberry Pi 5 installieren
 - 2. micro-RoS-Agent installieren
 - 2.2 Installation auf dem ESP32 mit PlatformIO
 - 1. PlatformIO in VS Code installieren
 - 2. PlatformIO-Umgebung konfigurieren
 - 3. micro-RoS für den ESP32 aktivieren
 - 3. ROS2-Package auf dem Raspberry Pi 5
 - 3.1 Erstellen eines ROS2-Pakets
 - Fazit

Nutzung von micro-RoS zwischen einem Raspberry Pi 5 und einem ESP32

1. Einführung in micro-RoS

Was ist micro-RoS?

micro-RoS ist eine angepasste Version von ROS2, die speziell für ressourcenbeschränkte Mikrocontroller entwickelt wurde. Sie erweitert das ROS2-Ökosystem um eingebettete Systeme, indem sie Kommunikation und Steuerung mit ROS2-kompatiblen Geräten ermöglicht.

Warum micro-RoS in der Robotik?

- **Integration in ROS2-Umgebungen:** Ermöglicht direkte Kommunikation zwischen einem Mikrocontroller und einem ROS2-System.
- **Optimierung für Embedded-Systeme:** Minimaler Speicherverbrauch und geringer CPU-Bedarf machen es ideal für ESP32, STM32 und andere Mikrocontroller.
- **Standardisierte Kommunikation:** Nutzt DDS (Data Distribution Service) zur Nachrichtenaustausch mit ROS2.

Architektur

Ein typisches Setup mit micro-RoS besteht aus:

1. **Raspberry Pi 5 (ROS2-Host):** Führt ROS2 aus und verwaltet die Kommunikation mit micro-RoS.
2. **ESP32 (micro-RoS-Agent):** Läuft auf einem Mikrocontroller und kommuniziert über **UART** oder **WiFi** mit ROS2.
3. **micro-RoS-Agent (auf dem Pi5):** Vermittelt Nachrichten zwischen dem ESP32 und ROS2.

2. Installation von micro-RoS auf Raspberry Pi 5 und ESP32

2.1 Installation auf dem Raspberry Pi 5

1. ROS2 auf dem Raspberry Pi 5 installieren

Falls ROS2 noch nicht installiert ist, installiere **ROS2 Jazzy**:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y ros-jazzy-ros-base
```

Source das ROS2-Setup:

```
echo "source /opt/ros/jazzy/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

2. micro-RoS-Agent installieren

```
sudo apt install python3-pip
pip install micro_ros_agent
```

Starte den micro-RoS-Agenten:

```
micro_ros_agent serial --dev /dev/ttyUSB0 -b 115200
```

Hinweis: Passe `/dev/ttyUSB0` ggf. an den Port des ESP32 an.

2.2 Installation auf dem ESP32 mit PlatformIO

1. PlatformIO in VS Code installieren

- Installiere **VS Code** und öffne die **PlatformIO-Erweiterung**.
- Erstelle ein neues PlatformIO-Projekt für den ESP32 mit dem **Arduino-Framework**.

2. PlatformIO-Umgebung konfigurieren

Bearbeite die Datei `platformio.ini` :

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200
lib_deps =
    micro-ROS/micro-ROS Arduino
    espressif/ArduinoJson
upload_port = /dev/ttyUSB0
monitor_port = /dev/ttyUSB0
upload_protocol = esptool
```

3. micro-ROS für den ESP32 aktivieren

Füge in der `src/main.cpp` eine minimale micro-ROS-Initialisierung hinzu:

```

#include <micro_ros_arduino.h>
#include <rcl/rcl.h>
#include <rcl/rclc.h>
#include <rclc/executor.h>
#include <std_msgs/msg/int32.h>

rcl_publisher_t publisher;
std_msgs__msg__Int32 msg;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rclc_executor_t executor;

void setup() {
  Serial.begin(115200);
  set_microros_transports();

  allocator = rcl_get_default_allocator();
  rclc_support_init(&support, 0, NULL, &allocator);
  rclc_node_init_default(&node, "esp32_node", "", &support);
  rclc_publisher_init_default(&publisher, &node, ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32), "motor_speed");

  msg.data = 0;
}

void loop() {
  msg.data = analogRead(34); // Simulierte Sensordaten
  rcl_publish(&publisher, &msg, NULL);
  delay(100);
}

```

3. ROS2-Package auf dem Raspberry Pi 5

3.1 Erstellen eines ROS2-Pakets

Wechsle in dein ROS2-Workspace:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python controller_pkg
```

Bearbeite controller_pkg/controller_pkg/gamepad_controller.py :

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Joy
from std_msgs.msg import Int32, Bool

class GamepadController(Node):
    def __init__(self):
        super().__init__('gamepad_controller')
        self.subscription = self.create_subscription(
            Joy, 'joy', self.joy_callback, 10)
        self.motor_pub = self.create_publisher(Int32, 'motor_speed', 10)
        self.led_pub_a = self.create_publisher(Bool, 'led_a', 10)
        self.led_pub_b = self.create_publisher(Bool, 'led_b', 10)

    def joy_callback(self, msg):
        speed = int(msg.axes[1] * 100) # Linker Joystick Y-Achse
        self.motor_pub.publish(Int32(data=speed))

        button_a = msg.buttons[0] # Button A
        button_b = msg.buttons[1] # Button B


        self.led_pub_a.publish(Bool(data=bool(button_a)))
        self.led_pub_b.publish(Bool(data=bool(button_b)))

def main(args=None):
    rclpy.init(args=args)
    node = GamepadController()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Fazit

Mit diesem Setup kann ein **Raspberry Pi 5** ROS2-Commands an einen **ESP32** senden, der **Motoren und LEDs** steuert. Der Gamepad-Controller ermöglicht dabei die einfache Steuerung.

 **ROS2 trifft Embedded-Systeme – Willkommen in der Zukunft der Robotik!**