Mecanum Robot with micro-ROS WiFi

Step 1: Wireless Control - Complete Guide

HARDWARE

• Arduino Portenta H7 + Mid Carrier
• 2x RoboClaw 2x15A Motor Controllers
• 4x goBILDA 5203 Motors (19.2:1, 312 RPM, 2150 CPR)
• 4x Mecanum Wheels (96mm diameter)
• 2x Matrix 12V 3000mAh NiMH Batteries

ROBOT DIMENSIONS

• Lx (wheel center-to-center width): 275mm
• Ly (wheel center-to-center length): 315mm
• Wheel diameter: 96mm

WIRING CONNECTIONS

RoboClaw #1 (Frontal - 0x80):

```
Portenta J14        RoboClaw #1
TX1  ■■■■■■■■■■■→    S1: I/O
RX1  ←■■■■■■■■■■■    S2: I/O
GND  ■■■■■■■■■■■→    S1: GND
```

RoboClaw #2 (Trasero - 0x81):

```
Portenta J14        RoboClaw #2
TX3  ■■■■■■■■■■■→    S1: I/O
RX3  ←■■■■■■■■■■■    S2: I/O
GND  ■■■■■■■■■■■→    S1: GND
```

Motor Distribution:

```
RoboClaw #1 (0x80): M1 = Front Right, M2 = Front Left
RoboClaw #2 (0x81): M1 = Rear Right,  M2 = Rear Left
```

ROBOCLAW CONFIGURATION (Motion Studio)

• RC1 Address: 128 (0x80)
• RC2 Address: 129 (0x81)
• Baudrate: 38400
• Mode: Packet Serial

WIFI CONFIGURATION

- SSID: PortentaROS
- Password: microros123
- Agent IP: 10.42.0.1
- UDP Port: 8888

SOFTWARE VERSIONS

- ROS2: Humble
- micro-ROS: humble
- PlatformIO Framework: Arduino
- Platform: ststm32

PORTENTA CODE

File: platformio.ini

```
[env:portenta_h7_m7]
platform = ststm32
board = portenta_h7_m7
framework = arduino

board_microros_transport = wifi
board_microros_distro = humble

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

File: src/main.cpp

```
#include <Arduino.h>
#include <micro_ros_platformio.h>
#include <rcl/rcl.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <geometry_msgs/msg/twist.h>

// WiFi
IPAddress agent_ip(10, 42, 0, 1);
size_t agent_port = 8888;
char ssid[] = "PortentaROS";
char psk[] = "microros123";

// RoboClaw
#define RC1_SERIAL Serial1
#define RC2_SERIAL Serial3
#define ROBOCLAW_BAUD 38400
#define RC1_ADDRESS 0x80
#define RC2_ADDRESS 0x81

// Comandos
#define CMD_M1_FORWARD 0
#define CMD_M1_BACKWARD 1
#define CMD_M2_FORWARD 4
#define CMD_M2_BACKWARD 5

// Robot dimensiones
#define WHEEL_RADIUS 0.048
#define LX 0.275
#define LY 0.315
#define L ((LX + LY) / 2.0)

// micro-ROS
rcl_subscription_t subscriber;
geometry_msgs__msg__Twist twist_msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;

#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)){}}
```

```
void error_loop() {
    while(1) {
        digitalWrite(LEDR, LOW); delay(100);
        digitalWrite(LEDR, HIGH); delay(100);
    }
}

uint16_t crc16(uint8_t *packet, int len) {
    uint16_t crc = 0;
    for (int i = 0; i < len; i++) {
        crc ^= ((uint16_t)packet[i] << 8);
        for (int j = 0; j < 8; j++) {
            if (crc & 0x8000) crc = (crc << 1) ^ 0x1021;
            else crc <<= 1;
        }
    }
    return crc;
}

void sendMotorCommand(HardwareSerial &serial, uint8_t address,
                      uint8_t cmd, uint8_t value) {
    uint8_t packet[] = {address, cmd, value};
    uint16_t crc = crc16(packet, 3);
    serial.write(address);
    serial.write(cmd);
    serial.write(value);
    serial.write((crc >> 8) & 0xFF);
    serial.write(crc & 0xFF);
    serial.flush();
}

void setMotor(HardwareSerial &serial, uint8_t address,
              uint8_t cmdFwd, uint8_t cmdBwd, float velocity) {
    int speed = constrain(abs(velocity) * 127, 0, 127);
    if (velocity >= 0) {
        sendMotorCommand(serial, address, cmdFwd, speed);
    } else {
        sendMotorCommand(serial, address, cmdBwd, speed);
    }
}

void driveMotors(float vx, float vy, float omega) {
    float v_fl = (vy - vx - L * omega) / WHEEL_RADIUS;
    float v_fr = (vy + vx + L * omega) / WHEEL_RADIUS;
    float v_rl = (vy + vx - L * omega) / WHEEL_RADIUS;
    float v_rr = (vy - vx + L * omega) / WHEEL_RADIUS;

    float max_speed = 10.0;
    float max_val = max(max(abs(v_fl), abs(v_fr)),
                        max(abs(v_rl), abs(v_rr)));
    if (max_val > max_speed) {
        v_fl = v_fl / max_val * max_speed;
        v_fr = v_fr / max_val * max_speed;
        v_rl = v_rl / max_val * max_speed;
        v_rr = v_rr / max_val * max_speed;
    }

    v_fl /= max_speed; v_fr /= max_speed;
    v_rl /= max_speed; v_rr /= max_speed;

    setMotor(RC1_SERIAL, RC1_ADDRESS, CMD_M2_FORWARD, CMD_M2_BACKWARD, v_fl);
    setMotor(RC1_SERIAL, RC1_ADDRESS, CMD_M1_FORWARD, CMD_M1_BACKWARD, v_fr);
    setMotor(RC2_SERIAL, RC2_ADDRESS, CMD_M2_FORWARD, CMD_M2_BACKWARD, v_rl);
    setMotor(RC2_SERIAL, RC2_ADDRESS, CMD_M1_FORWARD, CMD_M1_BACKWARD, v_rr);
}

void cmd_vel_callback(const void *msgin) {
    const geometry_msgs__msg__Twist *msg =
        (const geometry_msgs__msg__Twist *)msgin;
    float vx = msg->linear.y;
    float vy = msg->linear.x;
    float omega = msg->angular.z;
    driveMotors(vx, vy, omega);
    digitalWrite(LEDB, !digitalRead(LEDB));
}

void setup() {
    pinMode(LEDR, OUTPUT); pinMode(LEDG, OUTPUT); pinMode(LEDB, OUTPUT);
    digitalWrite(LEDR, HIGH); digitalWrite(LEDG, HIGH); digitalWrite(LEDB, HIGH);

    RC1_SERIAL.begin(ROBOCLAW_BAUD);
    RC2_SERIAL.begin(ROBOCLAW_BAUD);
    delay(100);

    set_microros_wifi_transports(ssid, psk, agent_ip, agent_port);
    delay(2000);

    allocator = rcl_get_default_allocator();
    RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
    RCCHECK(rclc_node_init_default(&node, "mecanum_robot", "", &support));
    RCCHECK(rclc_subscription_init_default(&subscriber, &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Twist), "cmd_vel"));
    RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber,
        &twist_msg, &cmd_vel_callback, ON_NEW_DATA));

    digitalWrite(LEDG, LOW);
}

void loop() {
    RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(1)));
}
```

PC TELEOP SCRIPT

File: teleop_qweasdzxc.py (save on Desktop)

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
import sys
import termios
import tty

class TeleopQWEASDZXC(Node):
    def __init__(self):
        super().__init__('teleop_qweasdzxc')
        self.publisher = self.create_publisher(Twist, 'cmd_vel', 10)

        self.speed = 0.15       # m/s (adjust as needed)
        self.turn_speed = 0.3   # rad/s (adjust as needed)

        self.key_map = {
            'w': ( 1,  0,  0),   # forward
            'x': (-1,  0,  0),   # backward
            'a': ( 0,  1,  0),   # lateral left
            'd': ( 0, -1,  0),   # lateral right
            'q': ( 1,  1,  0),   # diagonal front-left
            'e': ( 1, -1,  0),   # diagonal front-right
            'z': (-1,  1,  0),   # diagonal back-left
            'c': (-1, -1,  0),   # diagonal back-right
            'r': ( 0,  0,  1),   # rotate left
            't': ( 0,  0, -1),   # rotate right
            's': ( 0,  0,  0),   # stop
        }

        print("=== MECANUM CONTROL ===")
        print("   Q   W   E")
        print("   A   S   D")
        print("   Z   X   C")
        print("   R=rotate left  T=rotate right")
        print("   S=STOP  Ctrl+C=exit")
        print("=======================")

    def get_key(self):
        fd = sys.stdin.fileno()
        old = termios.tcgetattr(fd)
        try:
            tty.setraw(fd)
            key = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old)
        return key

    def run(self):
        try:
            while True:
                key = self.get_key().lower()
                if key == '\x03': break

                twist = Twist()
                if key in self.key_map:
                    vy, vx, omega = self.key_map[key]
                    twist.linear.x = float(vy * self.speed)
                    twist.linear.y = float(vx * self.speed)
                    twist.angular.z = float(omega * self.turn_speed)
                    print(f"Key: {key} -> vx:{twist.linear.x:.1f} "
                          f"vy:{twist.linear.y:.1f} w:{twist.angular.z:.1f}")
                self.publisher.publish(twist)
        finally:
            twist = Twist()
            self.publisher.publish(twist)

def main():
    rclpy.init()
    node = TeleopQWEASDZXC()
    node.run()
    node.destroy_node()
    rclpy.shutdown()

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

EXECUTION STEPS

Terminal 1 - Activate Hotspot:

```
nmcli connection up Hotspot
```

Terminal 2 - Start micro-ROS Agent:

```
docker run -it --rm --net=host microros/micro-ros-agent:humble udp4 --port 8888 -v4
```

Terminal 3 - Run Teleop:

```
cd ~/Desktop
source /opt/ros/humble/setup.bash
python3 teleop_qweasdzxc.py
```

Portenta:

Press the RESET button on the Portenta board.

Wait for agent to show 'session established', then use keyboard to control.

KEYBOARD CONTROLS

```
      Q              W              E
(diagonal left) (forward)   (diagonal right)

      A              S              D
(lateral left)   (STOP)     (lateral right)

      Z              X              C
(diag back-left) (backward) (diag back-right)

      R = rotate left      T = rotate right

      Ctrl+C = exit
```

SPEED ADJUSTMENT

Edit teleop_qweasdzxc.py and modify these values:

```
self.speed = 0.15       # linear speed (m/s)
self.turn_speed = 0.3   # angular speed (rad/s)
```

LED INDICATORS (Portenta)

• Green ON: Connected to micro-ROS agent

• Blue Blinking: Receiving cmd_vel commands
• Red Blinking: Error (check connections)

NEXT STEP

Step 2: Odometry - Use encoders to calculate robot position (x, y, θ) and publish to /odom topic for SLAM and autonomous navigation.