Mecanum Robot with micro-ROS

Technical Documentation - Step 1: Wireless Control

Project Overview

A 4-wheel omnidirectional mobile robot controlled wirelessly via micro-ROS and ROS2 Humble, running on an Arduino Portenta H7 microcontroller. This project demonstrates efficient embedded robotics without GPU or single-board computers, reducing power consumption and cost while maintaining full ROS2 integration.

Key Features

• Omnidirectional movement (forward, lateral, diagonal, rotation)
• Wireless control via micro-ROS over WiFi (UDP)
• No GPU/SBC - microcontroller only
• Full ROS2 Humble integration
• Encoder-ready for odometry (Step 2)

Hardware Specifications

Main Controller

Component	Specification
Board	Arduino Portenta H7
Carrier	Portenta Mid Carrier (ASX00055)
Processor	STM32H747 Dual-core (Cortex-M7 + M4)
Power	USB-C (5V)

Motor Controllers

Parameter	RoboClaw #1 (Front)	RoboClaw #2 (Rear)
Model	RoboClaw 2x15A	RoboClaw 2x15A
Address	0x80 (128)	0x81 (129)
UART	Serial1 (TX1/RX1)	Serial3 (TX3/RX3)
Baudrate	38400	38400
Mode	Packet Serial	Packet Serial
Motors	M1: Front Right, M2: Front Left	M1: Rear Right, M2: Rear Left

Motors and Encoders

Parameter	Value
Model	goBILDA 5203 Series Yellow Jacket
Gear Ratio	19.2:1
RPM	312

Encoder CPR	2150 counts per revolution
Quantity	4

Wheels and Power

Component	Specification
Wheels	Mecanum wheels, 96mm diameter
Batteries	2x Matrix 12V 3000mAh NiMH
Power Distribution	1 battery per RoboClaw

Robot Dimensions

Parameter	Symbol	Value
Wheel center-to-center (width)	Lx	275 mm
Wheel center-to-center (length)	Ly	315 mm
Wheel diameter	d	96 mm
Wheel radius	r	48 mm (0.048 m)
Kinematic parameter	L = (Lx+Ly)/2	295 mm (0.295 m)

Chassis Layout

```
                    FRONT
      +-------------------------------+
      |                               |
      |   [FL]                 [FR]   |
      |    M2                   M1    |
      |     \                   /     |
      |      \                 /      |
      |       +-------+-------+       |
      |       |       |       |       |
      |<-Lx-> |  275mm        |       |
      |       |       |       |       |
      |       +-------+-------+       |
      |      /        |Ly=315mm       |
      |     /                 \       |
      |    M2                   M1    |
      |   [RL]                 [RR]   |
      |                               |
      +-------------------------------+
                    REAR
```

```
[FL] Front Left  = RoboClaw #1 (0x80) M2
[FR] Front Right = RoboClaw #1 (0x80) M1
[RL] Rear Left   = RoboClaw #2 (0x81) M2
[RR] Rear Right  = RoboClaw #2 (0x81) M1
```

Mecanum Kinematics

Mecanum wheels have rollers mounted at 45° angles, enabling omnidirectional movement. Each wheel's velocity is calculated from the desired robot velocity:

```
Robot velocities:
    Vx = lateral velocity (m/s)
    Vy = forward velocity (m/s)
    ω  = angular velocity (rad/s)

Wheel angular velocities (rad/s):
    V_FL = (1/r) * (Vy - Vx - L*ω)
    V_FR = (1/r) * (Vy + Vx + L*ω)
    V_RL = (1/r) * (Vy + Vx - L*ω)
    V_RR = (1/r) * (Vy - Vx + L*ω)

Where:
    r = wheel radius = 0.048 m
    L = (Lx + Ly) / 2 = 0.295 m
```

Wiring Connections

RoboClaw #1 (Front) - Address 0x80

```
Portenta J14              RoboClaw #1
```

```
■■■■■■■■■■■■              ■■■■■■■■■■■
TX1  ■■■■■■■■■■■■■■■■■■   S1: I/O  (signal in)
RX1  ■■■■■■■■■■■■■■■■■■   S2: I/O  (signal out)
GND  ■■■■■■■■■■■■■■■■■■   S1: GND  (common ground)
```

RoboClaw #2 (Rear) - Address 0x81

```
Portenta J14              RoboClaw #2
■■■■■■■■■■■■              ■■■■■■■■■■■
TX3  ■■■■■■■■■■■■■■■■■■   S1: I/O  (signal in)
RX3  ■■■■■■■■■■■■■■■■■■   S2: I/O  (signal out)
GND  ■■■■■■■■■■■■■■■■■■   S1: GND  (common ground)
```

RoboClaw S1/S2 Header Pinout (left to right)

```
[ I/O ]  [ PWR ]  [ GND ]
   ■        ■        ■
 EDGE     CENTER   INTERIOR
(signal)  (+5V)    (ground)
```

Software Configuration

Software Versions

Component	Version
ROS2	Humble Hawksbill
micro-ROS	humble
PlatformIO Platform	ststm32
PlatformIO Framework	Arduino
Board	portenta_h7_m7
Transport	WiFi (UDP)

WiFi Configuration

Parameter	Value
SSID	PortentaROS
Password	microros123
Agent IP	10.42.0.1
UDP Port	8888
Host Interface	wlp4s0 (Ubuntu hotspot)

RoboClaw Configuration (Motion Studio)

Both RoboClaws must be configured with these settings:

Setting	RoboClaw #1	RoboClaw #2
Address	128 (0x80)	129 (0x81)
Baudrate	38400	38400
Mode	Packet Serial	Packet Serial
Multi-Unit Mode	Enabled	Enabled

Portenta LED Indicators

LED State	Meaning
Green ON	Connected to micro-ROS agent
Blue Blinking	Receiving cmd_vel commands
Red Blinking	Error - check connections

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

// RoboClaw Configuration
#define RC1_SERIAL Serial1  // TX1/RX1 - Front
#define RC2_SERIAL Serial3  // TX3/RX3 - Rear
#define ROBOCLAW_BAUD 38400
#define RC1_ADDRESS 0x80
#define RC2_ADDRESS 0x81

// RoboClaw Commands
#define CMD_M1_FORWARD 0
#define CMD_M1_BACKWARD 1
#define CMD_M2_FORWARD 4
#define CMD_M2_BACKWARD 5

// Robot Dimensions
#define WHEEL_RADIUS 0.048    // 96mm / 2
#define LX 0.275              // wheel center-to-center width
#define LY 0.315              // wheel center-to-center length
#define L ((LX + LY) / 2.0)   // 0.295m

// micro-ROS variables
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) {
    // Mecanum kinematics - calculate wheel velocities
    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;

    // Normalize if exceeding maximum
    float max_speed = 10.0;  // rad/s max
    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;
    }

    // Normalize to 0-1 range
    v_fl /= max_speed; v_fr /= max_speed;
    v_rl /= max_speed; v_rr /= max_speed;

    // Send to motors
    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;   // lateral
    float vy = msg->linear.x;   // forward
    float omega = msg->angular.z;  // rotation
    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);  // Green = connected
}

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

File: teleop_qweasdzxc.py (Host PC)

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

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 linear speed
        self.turn_speed = 0.3   # rad/s angular speed

        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 ROBOT TELEOP ===")
        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, timeout=0.1):
        fd = sys.stdin.fileno()
        old = termios.tcgetattr(fd)
        try:
            tty.setraw(fd)
            rlist, _, _ = select.select([sys.stdin], [], [], timeout)
            if rlist:
                key = sys.stdin.read(1)
            else:
                key = ''
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old)
        return key

    def run(self):
        twist = Twist()
        try:
            while True:
                key = self.get_key(0.05).lower()

                if key == '\x03':  # Ctrl+C
                    break

                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)
                elif key == '':
                    # No key pressed - stop
                    twist.linear.x = 0.0
                    twist.linear.y = 0.0
                    twist.angular.z = 0.0

                self.publisher.publish(twist)

        finally:
            # Stop on exit
            twist = Twist()
            self.publisher.publish(twist)

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

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

Keyboard Control Layout

```
        Q                W                E
   (diagonal ■)     (forward ↑)     (diagonal ■)

        A                S                D
   (lateral ←)       (STOP)        (lateral →)

        Z                X                C
   (diagonal ■)     (backward ↓)   (diagonal ■)

        R = rotate left (CCW)
        T = rotate right (CW)

        Ctrl+C = exit program
```

Execution Steps

Step 1: Activate WiFi Hotspot (Terminal 1)

```
nmcli connection up Hotspot
```

Step 2: Start micro-ROS Agent (Terminal 2)

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

Step 3: Run Teleop Script (Terminal 3)

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

Step 4: Reset Portenta

Press the RESET button on the Portenta board. Wait for the agent to display 'session established', then use the keyboard to control the robot.

Verification Commands

```
# Check if node is connected
ros2 node list
# Expected: /mecanum_robot

# Check available topics
ros2 topic list
# Expected: /cmd_vel

# Monitor cmd_vel messages
ros2 topic echo /cmd_vel
```

PlatformIO Commands

```
# Build firmware
pio run

# Upload to Portenta
pio run -t upload

# Open serial monitor
pio device monitor

# Clean micro-ROS library (after config changes)
pio run -t clean_microros
```

Next Steps

Step 2: Odometry

Implement odometry using encoder feedback to calculate and publish robot position (x, y, θ) on the /odom topic. This enables:
• Robot localization
• SLAM integration
• Autonomous navigation with Nav2

■■
■
Documentation Version: 1.0 | Step 1: Wireless Control
Repository: github.com/[your-username]/mecanum-robot