Mecanum Robot

Step 4: RViz Visualization with IMU + Joint States

Platform: Arduino Portenta H7 + micro-ROS

ROS Version: ROS2 Humble

Features: IMU Fusion, Odometry, Joint States, Real-time Visualization

Date: February 2026

Table of Contents

1. Overview & Features

Step 4 implements complete visualization of the mecanum robot in RViz with real-time odometry, IMU fusion for heading correction, and animated wheel joints. This step builds the foundation for autonomous navigation by establishing accurate state estimation and visualization capabilities.

Key Features Implemented:

• **Mecanum Odometry:** 4-wheel encoder-based position tracking with inverse kinematics

• **IMU Integration:** SparkFun ISM330DHCX for gyroscope-based heading correction

• **Joint States:** Real-time wheel position and velocity publishing

• **TF Tree:** Complete transform tree (odom → base_link → wheels, imu_link)

• **RViz Visualization:** 3D robot model with spinning wheels during movement

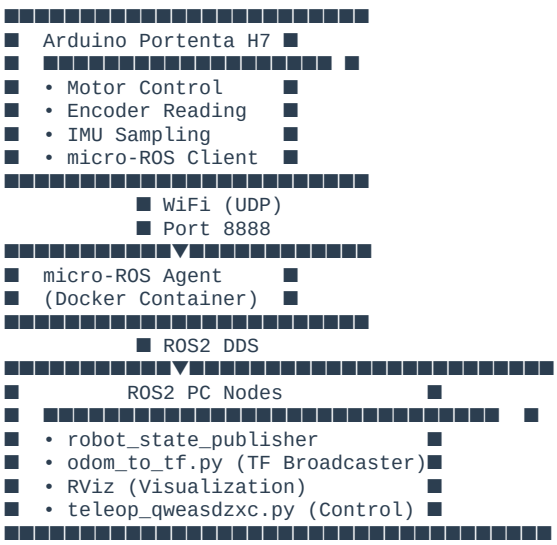
• **Teleop Control:** QWEASDZXC keyboard layout for omnidirectional control

Note: This step does NOT include sensor fusion via EKF. Raw odometry and IMU data are published separately. EKF integration will be implemented in Step 5.

2. System Architecture

The system uses micro-ROS over WiFi to connect the Arduino Portenta H7 with a ROS2 PC. Communication flows through a micro-ROS agent running in Docker.

Data Flow Diagram:



ROS2 Topics:

Topic	Type	Direction	Description
/cmd_vel	geometry_msgs/Twist	PC → Portenta	Velocity commands (vx, vy, ω)
/odom	nav_msgs/Odometry	Portenta → PC	Robot pose and velocity
/imu	sensor_msgs/Imu	Portenta → PC	IMU acceleration & gyro data
/joint_states	sensor_msgs/JointState	Portenta → PC	Wheel positions & velocities
/tf	tf2_msgs/TFMessage	PC	Transform tree
/robot_description	String (param)	PC	URDF model

3. Hardware Requirements

3.1 Robot Components

Component	Specification	Notes
Microcontroller	Arduino Portenta H7	Dual-core STM32H747, WiFi capable
Motor Controllers	2x RoboClaw (addresses 0x80, 0x81)	38400 baud, CRC enabled
Motors	4x DC motors with encoders	2150 CPR
Wheels	4x Mecanum wheels, 96mm diameter	0.048m radius
IMU	SparkFun ISM330DHCX	I2C address: 0x6B, 104Hz sample rate
Power	12V battery system	Separate 5V regulator for Portenta

3.2 Robot Physical Specifications

Parameter	Value	Description
Wheelbase (LX)	0.275 m	Front-rear distance
Track Width (LY)	0.315 m	Left-right distance
Wheel Radius	0.048 m	96mm diameter wheels
Encoder CPR	2150	Counts per revolution
Max Speed	~0.5 m/s	Typical operating speed

3.3 Pin Connections

• **RoboClaw 1 (0x80):** Serial1 (D14=TX, D13=RX) - Controls FR (M1) and FL (M2) wheels

• **RoboClaw 2 (0x81):** Serial3 (D16=TX, D17=RX) - Controls RR (M1) and RL (M2) wheels

• **IMU ISM330DHCX:** I2C (SDA=D14, SCL=D15) - Accelerometer and Gyroscope

4. Software Setup

4.1 Development Environment

The Portenta code uses PlatformIO for build management. Ensure you have PlatformIO installed in VS Code or use the CLI.

• **Arduino Framework:** PlatformIO with STM32 platform

• **micro-ROS:** micro_ros_platformio library (WiFi transport)

• **IMU Library:** SparkFun ISM330DHCX Arduino Library

4.2 ROS2 PC Requirements

• **OS:** Ubuntu 22.04 (or compatible)

• **ROS2:** Humble Hawksbill

• **Docker:** For micro-ROS agent

• **Python:** 3.10+ with rclpy

• **RViz2:** Included in ros-humble-desktop

4.3 Network Configuration

The Portenta connects to a WiFi hotspot created by the ROS2 PC. Configure your PC to create a hotspot named 'PortentaROS' with password 'microros123'.

```
# Create hotspot on Ubuntu
nmcli connection up Hotspot

# Verify IP assignment (should be 10.42.0.1)
ip addr show

# The Portenta will connect as a client and the micro-ROS
# agent listens on 10.42.0.1:8888
```

5. Portenta Code Implementation

5.1 platformio.ini

This configuration file sets up the build environment for micro-ROS on Portenta H7.

```
[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
    https://github.com/sparkfun/SparkFun_6DoF_ISM330DHCX_Arduino_Library.git
```

5.2 src/main.cpp - Complete Implementation

The main code is organized into several sections for clarity. Below is the complete implementation with detailed comments.

Part 1: Includes and Configuration

```
#include <Arduino.h>
#include <Wire.h>
#include <micro_ros_platformio.h>
#include <rcl/rcl.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <geometry_msgs/msg/twist.h>
#include <nav_msgs/msg/odometry.h>
#include <sensor_msgs/msg/imu.h>
#include <sensor_msgs/msg/joint_state.h>
#include <SparkFun_ISM330DHCX.h>

// WiFi Configuration
IPAddress agent_ip(10, 42, 0, 1);  // ROS2 PC hotspot IP
size_t agent_port = 8888;
char ssid[] = "PortentaROS";
char psk[] = "microros123";

// RoboClaw Serial Configuration
#define RC1_SERIAL Serial1  // RoboClaw 1: FR, FL
#define RC2_SERIAL Serial3  // RoboClaw 2: RR, RL
#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
#define CMD_GET_ENCODERS 78

// Robot Physical Dimensions
#define WHEEL_RADIUS 0.048          // meters
#define LX 0.275                    // wheelbase (m)
#define LY 0.315                    // track width (m)
#define L ((LX + LY) / 2.0)         // average for mecanum
#define ENCODER_CPR 2150.0          // counts per revolution
#define COUNTS_PER_METER (ENCODER_CPR / (2.0 * 3.14159 * WHEEL_RADIUS))

// IMU
SparkFun_ISM330DHCX imu;
#define DEG_TO_RAD 0.017453292519943
#define G_TO_MS2 9.80665

// micro-ROS Objects
rcl_subscription_t subscriber;
rcl_publisher_t odom_publisher;
rcl_publisher_t imu_publisher;
rcl_publisher_t joint_publisher;
geometry_msgs__msg__Twist twist_msg;
nav_msgs__msg__Odometry odom_msg;
sensor_msgs__msg__Imu imu_msg;
sensor_msgs__msg__JointState joint_msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t odom_timer;

// Joint state arrays
static char* joint_names[] = {
    "fl_wheel_joint", "fr_wheel_joint",
    "rl_wheel_joint", "rr_wheel_joint"
};
static double joint_positions[4] = {0, 0, 0, 0};
static double joint_velocities[4] = {0, 0, 0, 0};

// Odometry state
float x = 0.0, y = 0.0, theta = 0.0;
int32_t prev_fl = 0, prev_fr = 0, prev_rl = 0, prev_rr = 0;
```

```
int32_t total_fl = 0, total_fr = 0, total_rl = 0, total_rr = 0;
bool first_reading = true;
unsigned long last_odom_time = 0;

// IMU integration
float theta_imu = 0.0;
unsigned long last_imu_time = 0;

// Error handling macros
#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)){}}
```

Part 2: Error Handling and CRC

```
void error_loop() {
    // Flash red LED continuously on critical error
    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);
}
```

Part 3: Mecanum Drive Kinematics

```
void driveMotors(float vx, float vy, float omega) {
    // Mecanum inverse kinematics
    // vx: forward/backward velocity (m/s)
    // vy: left/right velocity (m/s)
    // omega: rotational velocity (rad/s)

    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;

    // Scale to maximum speed if necessary
    float max_speed = 10.0;  // rad/s
    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 [-1, 1] for motor commands
    v_fl /= max_speed;
    v_fr /= max_speed;
    v_rl /= max_speed;
    v_rr /= max_speed;

    // Send commands to RoboClaws
    setMotor(RC1_SERIAL, RC1_ADDRESS, CMD_M2_FORWARD,
             CMD_M2_BACKWARD, v_fl);  // FL
    setMotor(RC1_SERIAL, RC1_ADDRESS, CMD_M1_FORWARD,
             CMD_M1_BACKWARD, v_fr);  // FR
    setMotor(RC2_SERIAL, RC2_ADDRESS, CMD_M2_FORWARD,
             CMD_M2_BACKWARD, v_rl);  // RL
    setMotor(RC2_SERIAL, RC2_ADDRESS, CMD_M1_FORWARD,
             CMD_M1_BACKWARD, v_rr);  // RR
}

bool getEncoders(HardwareSerial &serial, uint8_t address,
                 int32_t &enc1, int32_t &enc2) {
    // Clear serial buffer
    while (serial.available()) serial.read();

    // Send command
    uint8_t cmd[] = {address, CMD_GET_ENCODERS};
    uint16_t crc = crc16(cmd, 2);
    serial.write(address);
    serial.write(CMD_GET_ENCODERS);
    serial.write((crc >> 8) & 0xFF);
    serial.write(crc & 0xFF);
    serial.flush();

    // Wait for response (10 bytes total)
    unsigned long start = millis();
    while (serial.available() < 10) {
        if (millis() - start > 50) return false;
    }

    // Read response
    uint8_t response[10];
    for (int i = 0; i < 10; i++) {
        response[i] = serial.read();
    }

    // Parse encoder values (32-bit signed integers)
    enc1 = ((int32_t)response[0] << 24) |
           ((int32_t)response[1] << 16) |
           ((int32_t)response[2] << 8) |
           response[3];
```

```
    enc2 = ((int32_t)response[4] << 24) |
           ((int32_t)response[5] << 16) |
           ((int32_t)response[6] << 8) |
           response[7];

    return true;
}
```

Part 4: IMU Update Function

```
void updateIMU() {
    sfe_ism_data_t accel, gyro;

    if (imu.checkAccelStatus() && imu.checkGyroStatus()) {
        imu.getAccel(&accel);
        imu.getGyro(&gyro);

        // Calculate dt
        unsigned long now = millis();
        float dt = (now - last_imu_time) / 1000.0;
        last_imu_time = now;

        // Integrate gyro Z for heading
        float gyro_z_rad = (gyro.zData / 1000.0) * DEG_TO_RAD;
        theta_imu += gyro_z_rad * dt;

        // Wrap angle to [-pi, pi]
        while (theta_imu > 3.14159) theta_imu -= 2 * 3.14159;
        while (theta_imu < -3.14159) theta_imu += 2 * 3.14159;

        // Populate IMU message
        imu_msg.header.stamp.sec = now / 1000;
        imu_msg.header.stamp.nanosec = (now % 1000) * 1000000;

        imu_msg.angular_velocity.x = (gyro.xData / 1000.0) * DEG_TO_RAD;
        imu_msg.angular_velocity.y = (gyro.yData / 1000.0) * DEG_TO_RAD;
        imu_msg.angular_velocity.z = gyro_z_rad;

        imu_msg.linear_acceleration.x = (accel.xData / 1000.0) * G_TO_MS2;
        imu_msg.linear_acceleration.y = (accel.yData / 1000.0) * G_TO_MS2;
        imu_msg.linear_acceleration.z = (accel.zData / 1000.0) * G_TO_MS2;

        // Publish
        RCSOFTCHECK(rcl_publish(&imu_publisher, &imu_msg, NULL));
    }
}
```

Part 5: Odometry Update Function

```
void updateOdometry() {
    // Read encoders from both RoboClaws
    int32_t fl, fr, rl, rr;
    if (!getEncoders(RC1_SERIAL, RC1_ADDRESS, fr, fl)) return;
    if (!getEncoders(RC2_SERIAL, RC2_ADDRESS, rr, rl)) return;

    // Initialize on first reading
    if (first_reading) {
        prev_fl = fl; prev_fr = fr;
        prev_rl = rl; prev_rr = rr;
        first_reading = false;
        last_odom_time = millis();
        last_imu_time = millis();
        return;
    }

    // Calculate encoder deltas
    int32_t d_fl = fl - prev_fl;
    int32_t d_fr = fr - prev_fr;
    int32_t d_rl = rl - prev_rl;
    int32_t d_rr = rr - prev_rr;

    // Update totals for absolute position
    total_fl += d_fl; total_fr += d_fr;
    total_rl += d_rl; total_rr += d_rr;

    // Update previous values
    prev_fl = fl; prev_fr = fr;
    prev_rl = rl; prev_rr = rr;

    // Convert encoder counts to distances
    float dist_fl = d_fl / COUNTS_PER_METER;
    float dist_fr = d_fr / COUNTS_PER_METER;
    float dist_rl = d_rl / COUNTS_PER_METER;
    float dist_rr = d_rr / COUNTS_PER_METER;

    // Mecanum forward kinematics
    float dx = (dist_fl + dist_fr + dist_rl + dist_rr) / 4.0;
    float dy = (-dist_fl + dist_fr + dist_rl - dist_rr) / 4.0;

    // Use IMU heading for orientation
    float avg_theta = theta_imu;

    // Update global position
    x += dx * cos(avg_theta) - dy * sin(avg_theta);
    y += dx * sin(avg_theta) + dy * cos(avg_theta);
    theta = theta_imu;

    // Calculate velocities
    unsigned long now = millis();
    float dt = (now - last_odom_time) / 1000.0;
    last_odom_time = now;

    float vx = (dt > 0) ? dx / dt : 0;
    float vy = (dt > 0) ? dy / dt : 0;
    float vtheta = imu_msg.angular_velocity.z;

    // Update joint states
    joint_positions[0] = (total_fl / ENCODER_CPR) * 2.0 * 3.14159;
    joint_positions[1] = (total_fr / ENCODER_CPR) * 2.0 * 3.14159;
    joint_positions[2] = (total_rl / ENCODER_CPR) * 2.0 * 3.14159;
    joint_positions[3] = (total_rr / ENCODER_CPR) * 2.0 * 3.14159;

    joint_velocities[0] = (dist_fl / WHEEL_RADIUS) / dt;
    joint_velocities[1] = (dist_fr / WHEEL_RADIUS) / dt;
    joint_velocities[2] = (dist_rl / WHEEL_RADIUS) / dt;
    joint_velocities[3] = (dist_rr / WHEEL_RADIUS) / dt;

    // Populate odometry message
    odom_msg.header.stamp.sec = now / 1000;
    odom_msg.header.stamp.nanosec = (now % 1000) * 1000000;
```

```
    odom_msg.pose.pose.position.x = x;
    odom_msg.pose.pose.position.y = y;
    odom_msg.pose.pose.position.z = 0.0;

    odom_msg.pose.pose.orientation.x = 0.0;
    odom_msg.pose.pose.orientation.y = 0.0;
    odom_msg.pose.pose.orientation.z = sin(theta / 2.0);
    odom_msg.pose.pose.orientation.w = cos(theta / 2.0);

    odom_msg.twist.twist.linear.x = vx;
    odom_msg.twist.twist.linear.y = vy;
    odom_msg.twist.twist.linear.z = 0.0;

    odom_msg.twist.twist.angular.x = 0.0;
    odom_msg.twist.twist.angular.y = 0.0;
    odom_msg.twist.twist.angular.z = vtheta;

    // Update joint message timestamp
    joint_msg.header.stamp.sec = now / 1000;
    joint_msg.header.stamp.nanosec = (now % 1000) * 1000000;
}
```

Part 6: Timer and Subscription Callbacks

```
void odom_timer_callback(rcl_timer_t *timer, int64_t last_call_time) {
    RCLC_UNUSED(last_call_time);
    if (timer != NULL) {
        // Update sensors and odometry
        updateIMU();
        updateOdometry();

        // Publish messages
        RCSOFTCHECK(rcl_publish(&odom_publisher, &odom_msg, NULL));
        RCSOFTCHECK(rcl_publish(&joint_publisher, &joint_msg, NULL));
    }
}

void cmd_vel_callback(const void *msgin) {
    const geometry_msgs__msg__Twist *msg =
        (const geometry_msgs__msg__Twist *)msgin;

    // Drive robot with received velocities
    driveMotors(msg->linear.y, msg->linear.x, msg->angular.z);

    // Toggle blue LED to indicate command received
    digitalWrite(LEDB, !digitalRead(LEDB));
}
```

Part 7: Setup Function

```
void setup() {
    // Initialize LEDs
    pinMode(LEDR, OUTPUT);
    pinMode(LEDG, OUTPUT);
    pinMode(LEDB, OUTPUT);
    digitalWrite(LEDR, HIGH);  // OFF
    digitalWrite(LEDG, HIGH);  // OFF
    digitalWrite(LEDB, HIGH);  // OFF

    // Initialize serial ports for RoboClaws
    RC1_SERIAL.begin(ROBOCLAW_BAUD);
    RC2_SERIAL.begin(ROBOCLAW_BAUD);
    delay(100);

    // Initialize I2C for IMU
    Wire.begin();
    if (!imu.begin(Wire, 0x6B)) {
        // Flash red LED if IMU initialization fails
        while(1) {
            digitalWrite(LEDR, LOW); delay(200);
            digitalWrite(LEDR, HIGH); delay(200);
        }
    }

    // Reset and configure IMU
    imu.deviceReset();
    while (!imu.getDeviceReset()) delay(1);

    imu.setAccelDataRate(ISM_XL_ODR_104Hz);
    imu.setGyroDataRate(ISM_GY_ODR_104Hz);

    // Initialize micro-ROS WiFi transport
    set_microros_wifi_transports(ssid, psk, agent_ip, agent_port);
    delay(2000);

    // Initialize micro-ROS
    allocator = rcl_get_default_allocator();
    RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
    RCCHECK(rclc_node_init_default(&node, "mecanum_robot", "", &support));

    // Initialize subscriber
    RCCHECK(rclc_subscription_init_default(
        &subscriber, &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Twist),
        "cmd_vel"));

    // Initialize publishers
    RCCHECK(rclc_publisher_init_default(
        &odom_publisher, &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(nav_msgs, msg, Odometry),
        "odom"));

    RCCHECK(rclc_publisher_init_default(
        &imu_publisher, &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, Imu),
        "imu"));

    RCCHECK(rclc_publisher_init_default(
        &joint_publisher, &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, JointState),
        "joint_states"));

    // Initialize timer (50ms = 20Hz)
    RCCHECK(rclc_timer_init_default(
        &odom_timer, &support,
        RCL_MS_TO_NS(50), odom_timer_callback));

    // Initialize executor
    RCCHECK(rclc_executor_init(&executor, &support.context, 2, &allocator));
    RCCHECK(rclc_executor_add_subscription(
        &executor, &subscriber, &twist_msg,
```

```
        &cmd_vel_callback, ON_NEW_DATA));
    RCCHECK(rclc_executor_add_timer(&executor, &odom_timer));

    // Initialize message frame IDs
    odom_msg.header.frame_id.data = (char*)"odom";
    odom_msg.header.frame_id.size = 4;
    odom_msg.header.frame_id.capacity = 5;

    odom_msg.child_frame_id.data = (char*)"base_link";
    odom_msg.child_frame_id.size = 9;
    odom_msg.child_frame_id.capacity = 10;

    imu_msg.header.frame_id.data = (char*)"imu_link";
    imu_msg.header.frame_id.size = 8;
    imu_msg.header.frame_id.capacity = 9;

    joint_msg.header.frame_id.data = (char*)"";
    joint_msg.header.frame_id.size = 0;
    joint_msg.header.frame_id.capacity = 1;

    // Initialize joint state message
    joint_msg.name.data =
        (rosidl_runtime_c__String*)malloc(4 * sizeof(rosidl_runtime_c__String));
    joint_msg.name.size = 4;
    joint_msg.name.capacity = 4;

    for (int i = 0; i < 4; i++) {
        joint_msg.name.data[i].data = joint_names[i];
        joint_msg.name.data[i].size = strlen(joint_names[i]);
        joint_msg.name.data[i].capacity = strlen(joint_names[i]) + 1;
    }

    joint_msg.position.data = joint_positions;
    joint_msg.position.size = 4;
    joint_msg.position.capacity = 4;

    joint_msg.velocity.data = joint_velocities;
    joint_msg.velocity.size = 4;
    joint_msg.velocity.capacity = 4;

    joint_msg.effort.data = NULL;
    joint_msg.effort.size = 0;
    joint_msg.effort.capacity = 0;

    // Green LED indicates successful initialization
    digitalWrite(LEDG, LOW);
}

void loop() {
    // Spin executor to process callbacks
    RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(1)));
}
```

6. ROS2 Workspace Setup

Create a ROS2 workspace for the robot description and Python scripts.

```
# Create workspace
mkdir -p ~/mecanum_viz_ws/src/mecanum_description
cd ~/mecanum_viz_ws/src/mecanum_description
mkdir urdf

# Files to create:
# - urdf/mecanum_robot.urdf
# - odom_to_tf.py
# - ~/Desktop/teleop_qweasdzxc.py
```

6.1 urdf/mecanum_robot.urdf

This URDF defines the robot structure with base_link, 4 wheels, and IMU.

```
<?xml version="1.0"?>
<robot name="mecanum_robot">
    <!-- Base Link -->
    <link name="base_link">
        <visual>
            <geometry>
                <box size="0.275 0.315 0.08"/>
            </geometry>
            <origin xyz="0 0 0.048" rpy="0 0 0"/>
            <material name="dark_blue">
                <color rgba="0.1 0.2 0.4 1"/>
            </material>
        </visual>
    </link>

    <!-- IMU Link -->
    <link name="imu_link">
        <visual>
            <geometry>
                <box size="0.025 0.025 0.005"/>
            </geometry>
            <material name="red">
                <color rgba="0.8 0.1 0.1 1"/>
            </material>
        </visual>
    </link>

    <joint name="imu_joint" type="fixed">
        <parent link="base_link"/>
        <child link="imu_link"/>
        <origin xyz="0 0 0.09" rpy="0 0 0"/>
    </joint>

    <!-- Front Left Wheel -->
    <link name="fl_wheel">
        <visual>
            <geometry>
                <cylinder radius="0.048" length="0.05"/>
            </geometry>
            <origin xyz="0 0 0" rpy="1.5708 0 0"/>
            <material name="orange">
                <color rgba="0.9 0.5 0.1 1"/>
            </material>
        </visual>
    </link>

    <joint name="fl_wheel_joint" type="continuous">
```

```
        <parent link="base_link"/>
        <child link="fl_wheel"/>
        <origin xyz="0.1375 0.1825 0" rpy="0 0 0"/>
        <axis xyz="0 1 0"/>
    </joint>

    <!-- Front Right Wheel -->
    <link name="fr_wheel">
        <visual>
            <geometry>
                <cylinder radius="0.048" length="0.05"/>
            </geometry>
            <origin xyz="0 0 0" rpy="1.5708 0 0"/>
            <material name="orange">
                <color rgba="0.9 0.5 0.1 1"/>
            </material>
        </visual>
    </link>

    <joint name="fr_wheel_joint" type="continuous">
        <parent link="base_link"/>
        <child link="fr_wheel"/>
        <origin xyz="0.1375 -0.1825 0" rpy="0 0 0"/>
        <axis xyz="0 1 0"/>
    </joint>

    <!-- Rear Left Wheel -->
    <link name="rl_wheel">
        <visual>
            <geometry>
                <cylinder radius="0.048" length="0.05"/>
            </geometry>
            <origin xyz="0 0 0" rpy="1.5708 0 0"/>
            <material name="orange">
                <color rgba="0.9 0.5 0.1 1"/>
            </material>
        </visual>
    </link>

    <joint name="rl_wheel_joint" type="continuous">
        <parent link="base_link"/>
        <child link="rl_wheel"/>
        <origin xyz="-0.1375 0.1825 0" rpy="0 0 0"/>
        <axis xyz="0 1 0"/>
    </joint>

    <!-- Rear Right Wheel -->
    <link name="rr_wheel">
        <visual>
            <geometry>
                <cylinder radius="0.048" length="0.05"/>
            </geometry>
            <origin xyz="0 0 0" rpy="1.5708 0 0"/>
            <material name="orange">
                <color rgba="0.9 0.5 0.1 1"/>
            </material>
        </visual>
    </link>

    <joint name="rr_wheel_joint" type="continuous">
        <parent link="base_link"/>
        <child link="rr_wheel"/>
        <origin xyz="-0.1375 -0.1825 0" rpy="0 0 0"/>
        <axis xyz="0 1 0"/>
    </joint>
</robot>
```

6.2 odom_to_tf.py

This script broadcasts TF transforms for odom→base_link and all wheel joints.

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from nav_msgs.msg import Odometry
from sensor_msgs.msg import JointState
from tf2_ros import TransformBroadcaster
from geometry_msgs.msg import TransformStamped
import math

class OdomAndJointToTF(Node):
    def __init__(self):
        super().__init__('odom_to_tf')
        self.br = TransformBroadcaster(self)
        self.sub_odom = self.create_subscription(
            Odometry, '/odom', self.odom_callback, 10)
        self.sub_joint = self.create_subscription(
            JointState, '/joint_states', self.joint_callback, 10)

    def odom_callback(self, msg):
        t = TransformStamped()
        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'odom'
        t.child_frame_id = 'base_link'
        t.transform.translation.x = msg.pose.pose.position.x
        t.transform.translation.y = msg.pose.pose.position.y
        t.transform.translation.z = msg.pose.pose.position.z
        t.transform.rotation = msg.pose.pose.orientation
        self.br.sendTransform(t)

    def joint_callback(self, msg):
        now = self.get_clock().now().to_msg()

        # Wheel positions from URDF
        wheel_configs = {
            'fl_wheel_joint': (0.1375, 0.1825, 0.0),
            'fr_wheel_joint': (0.1375, -0.1825, 0.0),
            'rl_wheel_joint': (-0.1375, 0.1825, 0.0),
            'rr_wheel_joint': (-0.1375, -0.1825, 0.0)
        }

        transforms = []
        for i, name in enumerate(msg.name):
            if name in wheel_configs:
                t = TransformStamped()
                t.header.stamp = now
                t.header.frame_id = 'base_link'
                t.child_frame_id = name.replace('_joint', '')

                pos = wheel_configs[name]
                t.transform.translation.x = pos[0]
                t.transform.translation.y = pos[1]
                t.transform.translation.z = pos[2]

                # Rotation around Y axis
                angle = msg.position[i]
                t.transform.rotation.x = 0.0
                t.transform.rotation.y = math.sin(angle / 2.0)
                t.transform.rotation.z = 0.0
                t.transform.rotation.w = math.cos(angle / 2.0)

                transforms.append(t)

        if transforms:
            self.br.sendTransform(transforms)

def main():
    rclpy.init()
    node = OdomAndJointToTF()
```

```
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

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

Important: Make the script executable:

```
chmod +x odom_to_tf.py
```

6.3 ~/Desktop/teleop_qweasdzxc.py

Keyboard teleop using QWEASDZXC layout for omnidirectional control.

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

class TeleopQWEASDZXC(Node):
    def __init__(self):
        super().__init__('teleop_qweasdzxc')
        self.publisher = self.create_publisher(Twist, 'cmd_vel', 10)
        self.speed = 0.15
        self.turn_speed = 0.3

        self.key_map = {
            'w': ( 1, 0, 0), 'x': (-1, 0, 0),  # Forward/Back
            'a': ( 0, 1, 0), 'd': ( 0, -1, 0),  # Left/Right
            'q': ( 1, 1, 0), 'e': ( 1, -1, 0),  # Diagonals
            'z': (-1, 1, 0), 'c': (-1, -1, 0),
            'r': ( 0, 0, 1), 't': ( 0, 0, -1),  # Rotate
            '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")

    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 == '':
                    twist.linear.x = 0.0
                    twist.linear.y = 0.0
                    twist.angular.z = 0.0

                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()
```

Important: Make the script executable:

```
chmod +x ~/Desktop/teleop_qweasdzxc.py
```

7. Execution Commands

Follow these steps in order to launch the complete system:

Step 1: Terminal 1 - WiFi Hotspot

```
nmcli connection up Hotspot
```

Step 2: Terminal 2 - micro-ROS Agent

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

Step 3: Portenta

```
Press RESET button after agent is running
```

Step 4: Terminal 3 - Teleop

```
cd ~/Desktop\nsource /opt/ros/humble/setup.bash\npython3 teleop_qweasdzxc.py
```

Step 5: Terminal 4 - Robot State Publisher

```
cd ~/mecanum_viz_ws\nsource install/setup.bash\nros2 run robot_state_publisher robot_state_publisher --ros-args -p robot
```

Step 6: Terminal 5 - TF Broadcaster

```
source /opt/ros/humble/setup.bash\npython3 ~/mecanum_viz_ws/src/mecanum_description/odom_to_tf.py
```

Step 7: Terminal 6 - RViz

```
source /opt/ros/humble/setup.bash\nQT_QPA_PLATFORM=xcb rviz2
```

In RViz:

• Set Fixed Frame to **odom**

• Add → RobotModel

• Add → TF (optional, for debugging)

• You should see the robot with orange wheels that spin when you drive

8. Troubleshooting Guide

Problem: Portenta LED blinks red continuously

Solution: IMU initialization failed. Check I2C connections and IMU address (0x6B).

Problem: No transform from [wheel] to [odom] in RViz

Solution: Ensure odom_to_tf.py is running and publishing transforms. Check: ros2 run tf2_ros tf2_echo odom base_link

Problem: Wheels visible but not spinning in RViz

Solution: Joint velocities may be zero. Verify /joint_states topic shows non-zero velocities when moving.

Problem: Robot not responding to teleop commands

Solution: Check /cmd_vel topic: ros2 topic echo /cmd_vel. Verify micro-ROS agent connection.

Problem: Odometry drifting significantly

Solution: Normal for raw odometry. EKF fusion (Step 5) will improve accuracy.

Problem: micro-ROS agent connection fails

Solution: Verify hotspot is active, IP is 10.42.0.1, and firewall allows UDP port 8888.

Useful Diagnostic Commands:

```
# Check topics
ros2 topic list

# Monitor odometry
ros2 topic echo /odom --once

# Monitor joint states
ros2 topic echo /joint_states --once

# Check TF tree
ros2 run tf2_ros tf2_echo odom base_link

# View TF tree
ros2 run rqt_tf_tree rqt_tf_tree

# Check node connectivity
ros2 node list
ros2 node info /mecanum_robot
```

9. Verification & Testing

9.1 System Health Checks

✓ Green LED on Portenta indicates successful micro-ROS connection

✓ Blue LED toggles when cmd_vel messages are received

✓ All topics publishing: /odom, /imu, /joint_states

✓ TF tree complete: odom → base_link → wheels, imu_link

✓ Robot model visible in RViz with correct orientation

✓ Wheels spin in RViz when robot moves

9.2 Movement Test Pattern

Test all mecanum degrees of freedom:

• **W:** Drive forward - verify positive X movement in RViz

• **X:** Drive backward - verify negative X movement

• **A:** Strafe left - verify positive Y movement

• **D:** Strafe right - verify negative Y movement

• **Q:** Diagonal forward-left - verify X and Y increase

• **E:** Diagonal forward-right - verify X increases, Y decreases

• **R:** Rotate counter-clockwise - verify orientation changes

• **T:** Rotate clockwise - verify orientation changes

• **S:** Stop - all motion ceases

9.3 Performance Metrics

Metric	Expected Value	Check Command
Odometry rate	~20 Hz	ros2 topic hz /odom
IMU rate	~20 Hz	ros2 topic hz /imu
Joint states rate	~20 Hz	ros2 topic hz /joint_states
TF update rate	~40 Hz	ros2 topic hz /tf
Cmd vel latency	<100 ms	Visual observation

10. Next Steps: EKF Integration (Step 5)

Step 4 provides raw sensor data. Step 5 will implement Extended Kalman Filter (EKF) sensor fusion to combine odometry and IMU data for improved pose estimation.

10.1 EKF Benefits

• **Improved Accuracy:** Fuses wheel odometry with IMU gyroscope data

• **Drift Reduction:** Corrects encoder slip and systematic errors

• **Covariance Estimation:** Provides uncertainty bounds for navigation

• **Standard Interface:** Publishes /odometry/filtered for Nav2 compatibility

10.2 EKF Configuration Overview

The robot_localization package provides EKF node that will:

• Subscribe to /odom (wheel odometry)

• Subscribe to /imu (IMU angular velocity)

• Publish /odometry/filtered (fused estimate)

• Broadcast odom → base_link TF (replaces odom_to_tf.py)

10.3 Preparation for Step 5

• Verify current system runs reliably for 5+ minutes

• Record baseline odometry drift over a known path

• Ensure IMU calibration is stable (check for gyro bias)

• Document any systematic errors (e.g., consistent turning bias)

Congratulations! You have completed Step 4. Your mecanum robot now has full visualization capabilities with real-time odometry, IMU integration, and spinning wheels in RViz. This establishes the foundation for autonomous navigation in Step 5.