Mecanum Robot - Step 2: Odometry + IMU

Complete Documentation

Overview

This step adds encoder-based odometry combined with IMU heading correction. The robot publishes position (x, y, theta) to /odom and raw IMU data to /imu topic at 20Hz.

Features

- Encoder odometry from 4 motors (2150 CPR)
- IMU heading correction using gyroscope Z-axis
- Publishing nav_msgs/Odometry to /odom
- Publishing sensor_msgs/Imu to /imu
- 20Hz update rate

IMU Configuration

```
IMU: SparkFun ISM330DHCX 6DOF
I2C Bus: Wire (I2C0)
I2C Address: 0x6B
Data Rate: 104Hz
Connection: Qwiic connector on Portenta H7

Pinout (Portenta H7 Qwiic):
  GND - Ground
  PH8 - I2C3 SDA (J1-44)
  PH7 - I2C3 SCL (J1-46)
  +5V - Power
```

ROS2 Topics

```
Published Topics:
  /odom  - nav_msgs/Odometry  - Robot position and velocity
  /imu   - sensor_msgs/Imu    - Raw IMU data (accel + gyro)

Subscribed Topics:
  /cmd_vel - geometry_msgs/Twist - Velocity commands
```

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

File: src/main.cpp

```
#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 <SparkFun_ISM330DHCX.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

// 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 dimensions
#define WHEEL_RADIUS 0.048
#define LX 0.275
#define LY 0.315
#define L ((LX + LY) / 2.0)
#define ENCODER_CPR 2150.0
#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
rcl_subscription_t subscriber;
rcl_publisher_t odom_publisher;
rcl_publisher_t imu_publisher;
geometry_msgs__msg__Twist twist_msg;
nav_msgs__msg__Odometry odom_msg;
sensor_msgs__msg__Imu imu_msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t odom_timer;

// Odometry variables
float x = 0.0, y = 0.0, theta = 0.0;
int32_t prev_fl = 0, prev_fr = 0, prev_rl = 0, prev_rr = 0;
bool first_reading = true;
unsigned long last_odom_time = 0;

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

#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);
}

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

    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();

    unsigned long start = millis();
    while (serial.available() < 10) {
        if (millis() - start > 50) return false;
    }

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

    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;
}

void updateIMU() {
    sfe_ism_data_t accel, gyro;

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

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

        // Integrate gyro Z for heading (convert from mdps to rad/s)
```

```
        float gyro_z_rad = (gyro.zData / 1000.0) * DEG_TO_RAD;
        theta_imu += gyro_z_rad * dt;

        // Normalize
        while (theta_imu > 3.14159) theta_imu -= 2 * 3.14159;
        while (theta_imu < -3.14159) theta_imu += 2 * 3.14159;

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

        // Angular velocity (mdps to rad/s)
        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;

        // Linear acceleration (mg to m/s^2)
        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;

        RCSOFTCHECK(rcl_publish(&imu_publisher, &imu_msg, NULL));
    }
}

void updateOdometry() {
    int32_t fl, fr, rl, rr;

    if (!getEncoders(RC1_SERIAL, RC1_ADDRESS, fr, fl)) return;
    if (!getEncoders(RC2_SERIAL, RC2_ADDRESS, rr, rl)) return;

    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;
    }

    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;

    prev_fl = fl; prev_fr = fr;
    prev_rl = rl; prev_rr = rr;

    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;

    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 for theta
    float avg_theta = theta_imu;
    x += dx * cos(avg_theta) - dy * sin(avg_theta);
    y += dx * sin(avg_theta) + dy * cos(avg_theta);
    theta = theta_imu;

    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;

    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;
}
```

```
void odom_timer_callback(rcl_timer_t *timer, int64_t last_call_time) {
    RCLC_UNUSED(last_call_time);
    if (timer != NULL) {
        updateIMU();
        updateOdometry();
        RCSOFTCHECK(rcl_publish(&odom_publisher, &odom_msg, NULL));
    }
}

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);

    Wire.begin();
    if (!imu.begin(Wire, 0x6B)) {
        while(1) {
            digitalWrite(LEDR, LOW); delay(200);
            digitalWrite(LEDR, HIGH); delay(200);
        }
    }
    imu.deviceReset();
    while (!imu.getDeviceReset()) delay(1);
    imu.setAccelDataRate(ISM_XL_ODR_104Hz);
    imu.setGyroDataRate(ISM_GY_ODR_104Hz);

    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_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_timer_init_default(&odom_timer, &support, RCL_MS_TO_NS(50), odom_timer_callback));

    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));

    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;

    digitalWrite(LEDG, LOW);
}

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
        self.turn_speed = 0.3

        self.key_map = {
            'w': ( 1,  0,  0),
            'x': (-1,  0,  0),
            'a': ( 0,  1,  0),
            'd': ( 0, -1,  0),
            'q': ( 1,  1,  0),
            'e': ( 1, -1,  0),
            'z': (-1,  1,  0),
            'c': (-1, -1,  0),
            'r': ( 0,  0,  1),
            't': ( 0,  0, -1),
            's': ( 0,  0,  0),
        }

        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, 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':
                    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()
```

Execution Steps

Terminal 1 - Activate Hotspot:

```
nmcli connection up Hotspot
```

Terminal 2 - micro-ROS Agent:

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

Terminal 3 - Teleop:

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

Terminal 4 - View Odometry:

```
source /opt/ros/humble/setup.bash
ros2 topic echo /odom
```

Terminal 5 - View IMU:

```
source /opt/ros/humble/setup.bash
ros2 topic echo /imu
```

Portenta:

Press RESET button after agent is running.

Configuration Reference

```
WiFi:
  SSID: PortentaROS
  Password: microros123
  Agent IP: 10.42.0.1
  Port: 8888

RoboClaw #1 (Front):
  Address: 0x80, UART: Serial1 (TX1/RX1)
  M1: Front Right, M2: Front Left

RoboClaw #2 (Rear):
  Address: 0x81, UART: Serial3 (TX3/RX3)
  M1: Rear Right, M2: Rear Left

IMU:
  Model: SparkFun ISM330DHCX 6DOF
  I2C Bus: Wire (I2C0)
  Address: 0x6B
  Data Rate: 104Hz

Robot Dimensions:
  Lx: 275mm, Ly: 315mm
  Wheel Diameter: 96mm
  Encoder CPR: 2150
```

LED Indicators

```
Green ON:        Connected to micro-ROS agent
Blue Blinking:   Receiving cmd_vel commands
Red Blinking:    Error (IMU not detected or connection issue)
```

Next Step: RViz Visualization

Step 3 will add RViz visualization with a 3D model of the robot, showing real-time position and orientation based on odometry and IMU data.