ChatGPT

ROS2 Motor Control Package for Raspberry Pi 5

Overview

This ROS 2 package (named **hsa_control**) allows a Raspberry Pi 5 to precisely control four DC motors (front-left, front-right, back-left, back-right) via two L298N dual H-bridge drivers. It subscribes to the `/motor_speeds` topic (message type `std_msgs/msg/Float32MultiArray`), expecting an array of four values (range –100 to 100) corresponding to the speed command for each motor [1]. Positive values make a motor spin forward, negative values make it spin in reverse, and the magnitude of the value sets the speed using PWM duty cycle (100 = 100% duty).

The node uses the RPi.GPIO library to interface with the Raspberry Pi's GPIO pins, which are wired to the L298N driver inputs as specified in the provided wiring diagram. Four hardware PWM-capable GPIO pins – BCM 18, 19, 12, and 13 – are used for the motor speed control (these connect to the ENA/ENB pins on the two L298N boards) [2]. Eight additional GPIO pins are used as digital outputs for motor direction control: BCM 17, 27, 22, 23 for the front motors (L298N board 1 IN1–IN4) [3], and BCM 5, 6, 20, 21 for the rear motors (L298N board 2 IN1–IN4) [4]. The software continuously updates the GPIO outputs to drive the motors according to the latest received speeds. If a new message is not received, the motors simply continue spinning at the last commanded speeds (until a stop command or shutdown).

Below are the contents of all necessary files for the ROS 2 package. You can copy these into a directory named **hsa_control** in your ROS 2 workspace (e.g. `ros2_ws/src/hsa_control/`). The package is structured for an **ament_python** build (pure Python), and includes a launch file for convenience.

Package Files

`package.xml`

```
<?xml version="1.0"?>
<package format="3">
  <name>hsa_control</name>
  <version>0.0.1</version>
  <description>ROS2 package for controlling four DC motors on Raspberry Pi 5
with L298N drivers.</description>
  <maintainer email="bingkun.huang@example.com">Bingkun Huang</maintainer>
  <license>MIT</license>

  <!-- Build tool dependencies -->
  <buildtool_depend>ament_cmake</buildtool_depend>
  <buildtool_depend>ament_cmake_python</buildtool_depend>

  <!-- Runtime dependencies (matches imports in code) -->
  <exec_depend>rclpy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
```

```
  <!-- (RPi.GPIO is a system library; ensure it's installed on the Pi) -->
</package>
```

Notes: This declares the package name, version, description, maintainer, and license. It specifies that we use the ROS 2 Python build system (`ament_cmake_python`) and that the code depends on the ROS 2 Python client library (`rclpy`) and standard messages (`std_msgs`) [5]. (The RPi.GPIO library is a standard Python module on Raspberry Pi OS, so it's not listed as a ROS dependency but should be installed on the system separately.)

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.8)
project(hsa_control)

find_package(ament_cmake REQUIRED)
find_package(ament_cmake_python REQUIRED)
find_package(rclpy REQUIRED)
find_package(std_msgs REQUIRED)

# Install Python modules
ament_python_install_package(${PROJECT_NAME})

# Install launch files
install(DIRECTORY launch
  DESTINATION share/${PROJECT_NAME}/
)

ament_package()
```

Notes: This CMake configuration enables the Python package build. It finds the required build dependencies and uses `ament_python_install_package` to install our Python module/package. The launch/ directory is also installed so that the launch file is available via `ros2 launch` [6]. We finish with `ament_package()` to finalize the build.

`setup.py`

```
from setuptools import setup, find_packages

package_name = 'hsa_control'

setup(
    name=package_name,
    version='0.0.1',
    packages=find_packages(include=[package_name]),
    data_files=[
        ('share/ament_index/resource_index/packages', ['resource/' +
package_name]),
        ('share/' + package_name, ['package.xml']),
        ('share/' + package_name + '/launch', ['launch/
```

```
motor_driver.launch.py']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='Bingkun Huang',
    maintainer_email='bingkun.huang@example.com',
    description='ROS2 node to control 4 DC motors on Raspberry Pi via L298N
drivers (subscribes to motor_speeds topic)',
    license='MIT',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            # Format: executable name = package.module:entry_point_function
            'motor_driver = hsa_control.motor_driver:main'
        ],
    },
)
```

Notes: This setup script uses setuptools to define the Python package. It installs the `hsa_control` module and makes sure the `package.xml` and launch file are included in the installation. The `entry_points` creates a console script called `motor_driver`, so that ROS 2 can execute our node using `ros2 run hsa_control motor_driver`. The maintainer, description, and license fields should match those in the `package.xml` [7].

`setup.cfg`

```
[develop]
script-dir=$base/lib/hsa_control
[install]
install-scripts=$base/lib/hsa_control
```

Notes: This configuration ensures that the installed executables (our console script) go into ROS 2's expected location (`lib/hsa_control`). This allows the `ros2 run` command to find the `motor_driver` script in the correct directory [8].

Python Node: `hsa_control/motor_driver.py`

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32MultiArray

# Import and configure Raspberry Pi GPIO library
import RPi.GPIO as GPIO

# Pin configuration (BCM numbering) matching the wiring diagram [2] [4]:
PWM_PINS = [18, 19, 12, 13]  # PWM pins: ENA1, ENB1, ENA2, ENB2 (front-left,
front-right, back-left, back-right)
DIR_PINS = [
```

```
    (17, 27),  # Direction pins for front-left motor (IN1-1, IN2-1)
    (22, 23),  # Direction pins for front-right motor (IN3-1, IN4-1)
    (5,  6),   # Direction pins for back-left motor (IN1-2, IN2-2)
    (20, 21)   # Direction pins for back-right motor (IN3-2, IN4-2)
]

class MotorControllerNode(Node):
    def __init__(self):
        super().__init__('motor_controller')
        # Initialize GPIO pins
        GPIO.setmode(GPIO.BCM)
        # Set all direction and PWM pins as outputs and initialize them LOW
        for pin in [pin for pair in DIR_PINS for pin in pair] + PWM_PINS:
            GPIO.setup(pin, GPIO.OUT, initial=GPIO.LOW)
        # Set up PWM on each PWM-capable pin at 1000 Hz, initially 0% duty
cycle
        self.pwms = [GPIO.PWM(p, 1000) for p in PWM_PINS]
        for pwm in self.pwms:
            pwm.start(0)

        # Create a subscriber to the motor_speeds topic
        self.subscription = self.create_subscription(
            Float32MultiArray,
            'motor_speeds',
            self.motor_speeds_callback,
            10  # QoS depth
        )

self.get_logger().info('MotorControllerNode initialized and listening on /
motor_speeds')

    def motor_speeds_callback(self, msg: Float32MultiArray):
        """Handle incoming motor speed commands."""
        speeds = msg.data
        if len(speeds) < 4:
            self.get_logger().error(f"Received motor_speeds array of length
{len(speeds)}; expected 4 values.")
            return
        # Loop through each motor and apply the speed command
        for idx, speed in enumerate(speeds[:4]):
            # Constrain speed to [-100, 100]
            if speed > 100.0:
                speed = 100.0
            if speed < -100.0:
                speed = -100.0

            # Determine direction and set direction pins
            forward = (speed >= 0.0)
            GPIO.output(DIR_PINS[idx][0], GPIO.HIGH if forward else GPIO.LOW)
            GPIO.output(DIR_PINS[idx][1], GPIO.LOW if forward else GPIO.HIGH)
```

```
            # Set PWM duty cycle to the magnitude of speed (as percentage)
            duty = abs(speed)
            self.pwms[idx].ChangeDutyCycle(duty)

# Optionally log the received speeds (commented out to avoid flooding logs)
        # self.get_logger().info(f'Applied motor speeds: {speeds}')

    def destroy_node(self):
        """Cleanup GPIO on shutdown."""
        # Stop PWM signals and clean up GPIO to release pins
        for pwm in self.pwms:
            pwm.stop()
        GPIO.cleanup()
        self.get_logger().info("GPIO cleanup complete.")
        super().destroy_node()

def main(args=None):
    rclpy.init(args=args)
    node = MotorControllerNode()
    try:
        rclpy.spin(node)   # Keep the node running to listen for messages
    except KeyboardInterrupt:
        node.get_logger().info("Motor controller node interrupted by user
(CTRL-C).")
    finally:
        # On shutdown, destroy the node (this triggers GPIO cleanup) and
shutdown rclpy
        if rclpy.ok():
            node.destroy_node()
        rclpy.shutdown()
```

Explanation: This is the Python ROS 2 node that controls the motors. It sets up the GPIO pins according to the wiring (BCM numbering). The four PWM pins [18, 19, 12, 13] correspond to the enable inputs of the L298N drivers (for front-left, front-right, back-left, back-right respectively) [2]. The `DIR_PINS` list contains tuples of two GPIO pins for each motor's direction inputs (for example, `(17, 27)` are the IN1 and IN2 pins for the front-left motor) [3] [4]. In the `MotorControllerNode` constructor, all these pins are set as outputs and initialized LOW, and a 1000 Hz PWM signal is prepared on each enable pin (initially 0% duty cycle, so motors are stopped). The node then creates a subscription to the `/motor_speeds` topic.

When a message arrives, the `motor_speeds_callback` will iterate through the first four values of the array and set each motor. For each motor index, it determines the direction (sets one GPIO high and the other low, or vice versa) and updates the PWM duty cycle according to the speed magnitude [1]. For example, if a speed value is negative, the code sets the first pin LOW and second pin HIGH to reverse the motor's polarity, and if positive, it does the opposite (first HIGH, second LOW). The duty cycle is set to `abs(speed)` (which corresponds to 0–100% duty). Any values outside the range [–100, 100] are clamped. If the array is shorter than 4 values, an error is logged and the message is ignored. The node logs a startup message and (optionally) can log applied speeds for debugging.

Importantly, the `destroy_node` method is overridden to stop all PWM signals and call `GPIO.cleanup()` when the node is shutting down. This ensures that the motors stop (and all GPIO resources are freed) when you exit the node. The `main` function initializes ROS, instantiates the node, and spins it. On program exit (Ctrl-C or otherwise), it catches the signal, stops the node, and cleans up properly.

Launch File: `launch/motor_driver.launch.py`

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='hsa_control',
            executable='motor_driver',
            name='motor_controller',
            output='screen'
        )
    ])
```

Notes: This launch file starts the motor controller node with a descriptive name and outputs its log to the screen. After building the package, you can use this file to run the node via the ROS 2 launch system (as shown below).

Building and Running the Package

To build and use this package on the Raspberry Pi (with Debian/ROS 2 installed), follow these steps:

1. **Install RPi.GPIO (if needed):** Ensure the RPi.GPIO Python library is available on your system. On Raspberry Pi OS, it is usually pre-installed. On Debian, you may need to install it via `sudo apt-get install python3-rpi.gpio` (or via pip). Also, make sure your user has permission to access GPIO (on Raspberry Pi OS, add your user to the `gpio` group or run the node with `sudo` if necessary).

2. **Add the package to your workspace:** Copy the **hsa_control** package directory (containing the files above) into the `src` folder of your ROS 2 workspace (e.g. `~/ros2_ws/src/hsa_control/`).

3. **Build the package:** Navigate to the root of your workspace (e.g. `~/ros2_ws`) and run:

   ```
   colcon build --packages-select hsa_control
   ```

   This will compile/install the package. Make sure to source the setup script after building:

   ```
   source install/setup.bash
   ```

4. **Launch the motor controller node:** You can launch the node using ROS 2 launch:

   ```
   ros2 launch hsa_control motor_driver.launch.py
   ```

   This will start the node (named "motor_controller") which begins listening on the `/motor_speeds` topic. You should see a log message indicating that it is initialized and listening.

5. **Publish motor speed commands:** Send commands to the motors by publishing to the `/motor_speeds` topic. For example, to test the motors, you can open a new terminal (with the ROS 2 environment sourced) and run:

   ```
   ros2 topic pub /motor_speeds std_msgs/msg/Float32MultiArray "data:
   [50.0, 50.0, 50.0, 50.0]"
   ```

   This command will publish an array of four speeds (each 50.0) to the topic, causing all four motors to spin forward at ~50% speed. You should observe the motors turning. You can try other values (e.g., negative values to reverse direction). The node will continuously apply the last received speeds until new commands arrive.

6. **Stopping the motors:** To stop the motors, you can publish zero speeds (`[0.0, 0.0, 0.0, 0.0]`) or simply shut down the node. When the node is terminated (e.g., by pressing Ctrl+C in the launch terminal), it will stop all PWM signals and release the GPIOs, effectively stopping all motors.

Summary: This package provides a ready-to-use ROS 2 solution for controlling four DC motors on a Raspberry Pi 5 with L298N drivers. It subscribes to a topic for commands, uses the wiring and control logic as described in the provided documentation [9] [1], and includes launch and config files to integrate into your ROS 2 system. After copying these files into the GitHub repository (under `HSA_control`) and building, you will have everything needed to run the motor controller as part of your ROS 2 application. Enjoy your ROS-powered robotic motor control setup!

Sources:

- Raspberry Pi 5 to L298N wiring and example control code [10] [4] [1] (reference for GPIO pin assignments and motor control logic)

---

[1] [2] [3] [4] [9] [10] e05b1058-9404-47f3-9db7-49da695310ca.pdf
file://file-5KeVuKkZbnss47xfhXfzgX

[5] [6] [7] [8] Writing a simple publisher and subscriber (Python) — ROS 2 Documentation: Foxy documentation
https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html