Documentation: Using the ROS PID-Controlled Motor Driver Code

Overview

This code integrates PID control, encoder feedback, and ROS communication to control two DC motors in a differential drive robot. The setup includes motor drivers, encoders, and a ROS interface to handle velocity commands (`cmd_vel`) and publish encoder feedback.

Features

- **PID Control**: Ensures smooth and precise control of motor speeds.
- **ROS Integration**: Receives velocity commands and publishes encoder data for feedback.
- **Encoder Feedback**: Provides real-time motor position and speed updates.
- **Motor Direction Control**: Handles forward, reverse, and turning movements.

Components and Pin Configuration

Motor and Direction Pins

- `MOTOR_PIN_1` (Pin 9): Controls PWM for the left motor.
- `MOTOR_PIN_2` (Pin 10): Controls PWM for the right motor.

- `DIR_PIN_1` (Pin 7): Direction control for the left motor.
- `DIR_PIN_2` (Pin 6): Direction control for the right motor.

Encoder Pins
- `LEFT_ENCODER_PIN_A` (Pin 18)
- `LEFT_ENCODER_PIN_B` (Pin 19)
- `RIGHT_ENCODER_PIN_A` (Pin 2)
- `RIGHT_ENCODER_PIN_B` (Pin 3)

Setup Instructions

Hardware
1. **Connect Motors**: Connect the motors to the motor driver and the motor driver to the microcontroller as per the pin configuration.
2. **Connect Encoders**: Attach the encoder outputs to the microcontroller pins.
3. **Power Supply**: Ensure sufficient power for motors and the microcontroller.

Software
1. **Install Arduino Libraries**:
   - `PID_v1` for PID control.
   - `rosserial_arduino` for ROS communication.
2. **ROS Setup**:
   - Ensure `rosserial` is installed on your ROS workspace.
   - Run the `rosserial` node to establish communication between ROS and the microcontroller.

Code Workflow

Initialization (`setup`)

1. Motor pins are set as outputs.
2. Encoder pins are configured as inputs with interrupts to handle encoder tick counts.
3. PID controllers are initialized with initial parameters.
4. ROS node is initialized to handle subscribers and publishers.

Main Loop (`loop`)

1. **Update PID Input**:
   - The encoder values (`left_encoder_ticks` and `right_encoder_ticks`) are used as inputs for the PID controller.
2. **Process ROS Callbacks**:
   - Handle incoming `cmd_vel` messages.
3. **Publish Encoder Values**:
   - Encoder values are sent back to the ROS master for monitoring or further processing.

Velocity Callback (`velCallback`)

1. **Process `cmd_vel`**:
   - Updates target velocities (`Setpoint1` and `Setpoint2`) based on incoming messages.
2. **Adjust PID Tunings**:
   - Switch between aggressive and conservative tunings based on the gap between setpoint and input.
3. **Compute and Apply Outputs**:

   - Calculate PID outputs and apply them to the motors to achieve the desired motion.

ROS Topics

Subscribed Topics
- `/cmd_vel` (`geometry_msgs/Twist`): Receives velocity commands for linear and angular motion.

Published Topics
- `/lwheel` (`std_msgs/Int16`): Publishes left wheel encoder ticks.
- `/rwheel` (`std_msgs/Int16`): Publishes right wheel encoder ticks.

Key Functions

`publishEncoders()`
- Publishes current encoder tick values to ROS topics.

`leftEncoderISR()` and `rightEncoderISR()`
- Interrupt Service Routines (ISRs) to update encoder tick counts based on encoder state changes.

`velCallback()`
- Handles incoming velocity commands to adjust motor speeds and directions.

Customization

PID Tuning
Modify PID parameters in the code for your specific motor and application:
- Aggressive Mode:
  ```cpp
  double aggKp = 4, aggKi = 0.2, aggKd = 1;
  ```
- Conservative Mode:
  ```cpp
  double consKp = 1, consKi = 0.05, consKd = 0.25;
  ```

ROS Topics
Adjust the topic names to match your ROS configuration:
```cpp
ros::Publisher left_wheel_pub("lwheel", &left_wheel_msg);
ros::Publisher right_wheel_pub("rwheel", &right_wheel_msg);
ros::Subscriber<geometry_msgs::Twist> vel_sub("cmd_vel", velCallback);
```

Testing the Code

1. **Start ROS Node**:
   - Run `rosserial` on your ROS master.

```bash
rosrun rosserial_python serial_node.py _port:=/dev/ttyUSB0 _baud:=115200
```

2. **Publish Commands**:
   - Send velocity commands via `rostopic` or a custom ROS node.
   ```bash
   rostopic pub /cmd_vel geometry_msgs/Twist '{linear: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.2}}'
   ```

3. **Monitor Feedback**:
   - Check encoder feedback topics `/lwheel` and `/rwheel`.

Troubleshooting

1. **No Movement**:
   - Check power connections to motors and the microcontroller.
   - Verify the motor driver connections and functionality.

2. **ROS Communication Issues**:
   - Ensure `rosserial` is correctly configured.
   - Verify the port and baud rate.

3. **Incorrect Direction**:
   - Swap motor wires or adjust the direction control logic.

This documentation provides a step-by-step guide to understanding and using the code effectively. Adapt the configurations and tunings as needed for your specific hardware and application.