Modern TurtleBot3 Teleop Controller - Setup and Usage Guide

Overview

This modern C++ teleop controller for TurtleBot3 includes advanced features like smooth velocity transitions, safety obstacle avoidance, real-time status display, and modern C++ best practices.

Features

🚀 Modern C++ Features

- **RAII (Resource Acquisition Is Initialization)**: Proper resource management
- **Smart Pointers**: Automatic memory management with `std::shared_ptr`
- **Atomic Variables**: Thread-safe operations for multi-threading
- **Lambdas**: Clean, inline function definitions
- **Range-based Loops**: Modern iteration patterns
- **Auto Type Deduction**: Cleaner code with automatic type inference
- **Constexpr**: Compile-time constants for better performance
- **Uniform Initialization**: Consistent initialization syntax

🛡️ Safety Features

- **Obstacle Avoidance**: Automatic stopping when obstacles detected
- **Auto-stop**: Robot stops after 2 seconds of no input
- **Velocity Limiting**: Prevents excessive speeds
- **Smooth Transitions**: Gradual acceleration/deceleration
- **Safety Toggle**: Enable/disable safety features on the fly

📊 Advanced Controls

- **Multiple Speed Modes**: Slow, Normal, Fast presets
- **Real-time Status Display**: Live velocity, position, and sensor data
- **Diagonal Movement**: 8-directional control (WASD + QE + ZC)
- **Dynamic Speed Adjustment**: Increase/decrease max speeds
- **Interactive Help**: Built-in command reference

Setup Instructions

1. File Structure

```
~/turtlebot3_ws/src/turtlebot3_custom/
├── src/
│   ├── modern_teleop_controller.cpp
│   ├── turtlebot3_controller.cpp
│   └── ... (other existing files)
├── launch/
│   ├── modern_teleop.launch.py
│   └── ... (other launch files)
├── CMakeLists.txt
└── package.xml
```

2. Build the Package

```bash
# Navigate to workspace
cd ~/turtlebot3_ws

# Build with the new teleop controller
colcon build --packages-select turtlebot3_custom

# Source the workspace
source install/setup.bash
```

3. Environment Setup

```bash
# Set TurtleBot3 model
export TURTLEBOT3_MODEL=burger

# For simulation
export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:/opt/ros/humble/share/turtlebot3_gazebo/models
```

Usage Instructions

Running in Simulation

Option 1: Complete Simulation Launch

```bash
# Terminal 1: Start everything (Gazebo + Teleop)
cd ~/turtlebot3_ws
source install/setup.bash
ros2 launch turtlebot3_custom modern_teleop.launch.py start_gazebo:=true use_sim_time:=true
```

Option 2: Step-by-Step Launch

```bash
# Terminal 1: Start Gazebo
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py

# Terminal 2: Start Teleop Controller
cd ~/turtlebot3_ws && source install/setup.bash
ros2 run turtlebot3_custom modern_teleop_controller
```

Running with Real Robot

```bash
# On TurtleBot3 (Raspberry Pi)
export TURTLEBOT3_MODEL=burger
export ROS_DOMAIN_ID=30
ros2 launch turtlebot3_bringup robot.launch.py

# On your PC
export TURTLEBOT3_MODEL=burger
export ROS_DOMAIN_ID=30
cd ~/turtlebot3_ws && source install/setup.bash
ros2 run turtlebot3_custom modern_teleop_controller
```

Control Reference

Basic Movement

Key	Action	Description
W	Forward	Move robot forward
S	Backward	Move robot backward
A	Left	Turn robot left
D	Right	Turn robot right

Key	Action	Description
X	Stop	Immediate stop

Diagonal Movement

Key	Action	Description
Q	Forward-Left	Move forward while turning left
E	Forward-Right	Move forward while turning right
Z	Backward-Left	Move backward while turning left
C	Backward-Right	Move backward while turning right

Speed Control

Key	Action	Description
1	Slow Mode	Reduced maximum speeds
2	Normal Mode	Standard TurtleBot3 speeds
3	Fast Mode	Increased maximum speeds
+/=	Increase Speed	Boost maximum speeds
-	Decrease Speed	Reduce maximum speeds

Safety & Utility

Key	Action	Description
T	Toggle Safety	Enable/disable obstacle avoidance
R	Reset	Reset all velocities to zero
H	Help	Show command reference
ESC	Quit	Exit teleop controller

Advanced Features Explained

1. Smooth Velocity Transitions

- Uses gradual acceleration/deceleration instead of instant velocity changes
- Prevents mechanical stress and improves control precision
- Implemented with lambda functions for clean code

2. Thread-Safe Operations

- Uses `std::atomic` for thread-safe variable access
- Separate thread for keyboard input processing
- Main control loop runs independently at 20Hz

3. Safety System

- **Obstacle Detection**: Uses laser scan data to detect obstacles within 30° forward arc
- **Distance Threshold**: Stops forward movement when obstacles closer than 0.3m
- **Auto-timeout**: Stops robot after 2 seconds of no input
- **Can be toggled**: Press 't' to enable/disable safety features

4. Real-time Status Display

- Live velocity display (current/target)
- Robot position and orientation
- Obstacle distance
- Safety mode status
- Updates every 500ms

Troubleshooting

Common Issues

1. Controller Not Responding

```bash
# Check if the node is running
ros2 node list | grep teleop

# Check if cmd_vel is being published
ros2 topic echo /cmd_vel
```

2. No Obstacle Detection

```bash
```

```
# Check if laser scan is available
ros2 topic echo /scan

# Verify laser scan frequency
ros2 topic hz /scan
```

3. Build Errors

```
bash

# Clean build
cd ~/turtlebot3_ws
rm -rf build/ install/ log/
colcon build --packages-select turtlebot3_custom
```

4. Terminal Input Issues

- Make sure terminal has focus when controlling
- If keys don't respond, press Ctrl+C to exit and restart
- Some terminals may require specific settings for raw input

Performance Monitoring

```
bash

# Check message rates
ros2 topic hz /cmd_vel
ros2 topic hz /scan
ros2 topic hz /odom

# Monitor CPU usage
top -p $(pgrep -f modern_teleop)

# View node graph
ros2 run rqt_graph rqt_graph
```

Code Architecture Highlights

Modern C++ Patterns Used

1. **RAII Pattern**: Terminal settings automatically restored in destructor
2. **Smart Pointers**: All ROS2 publishers/subscribers use shared_ptr

3. **Atomic Variables**: Thread-safe access to velocity and sensor data
4. **Lambda Functions**: Used for smooth velocity transitions and callbacks
5. **Const Correctness**: Proper const usage throughout
6. **Exception Safety**: Try-catch blocks for robust error handling
7. **STL Algorithms**: Using `std::clamp`, `std::min`, `std::copysign`

Threading Model

- **Main Thread**: ROS2 spin loop handling callbacks
- **Input Thread**: Dedicated keyboard input processing
- **Timer Callbacks**: Control loop (50ms) and status display (500ms)

Safety Architecture

- **Multi-layered Safety**: Obstacle detection + timeout + velocity limits
- **Fail-safe Design**: Default to safe state on errors
- **User Control**: Safety can be toggled for testing/debugging

This modern teleop controller demonstrates professional C++ development practices while providing a robust and feature-rich control interface for TurtleBot3!