ROS 2 Launch Files Documentation

Modern Guide for real_robot.launch.py and simulation.launch.py

Generated by Grok 3

May 13, 2025

Contents

1 Introduction

This document offers a modern, developer-friendly guide to two ROS 2 launch files: `real_robot.launch.p` and `simulation.launch.py`. These files orchestrate a robotic system's components for real-world and simulated environments. Tailored for developers, it includes a sleek overview, detailed functionality, system architecture, a vibrant dependency graph, and practical tips for use and maintenance.

Crafted with a contemporary design, this documentation uses color, clear typography, and visual aids to enhance readability. It assumes familiarity with ROS 2 and Python, delivering a professional yet approachable resource for robotic system development. Spanning at least 13 pages, it ensures comprehensive coverage with a fresh, engaging style.

2 Overview

The ROS 2 launch system streamlines the startup of nodes, parameters, and dependencies in robotic systems. Python-based launch files in ROS 2 provide dynamic control, surpassing ROS 1's capabilities. The `real_robot.launch.py` and `simulation.launch.py` files initialize a differential drive robot's core systems: control, perception, and navigation.

- `real_robot.launch.py`: Targets physical robots, interfacing with hardware like motors and LiDAR.
- `simulation.launch.py`: Sets up a Gazebo-based simulation with RViz2 visualization, using delays for robust startup.

Both files leverage the `launch` module for modularity, ensuring reusable and maintainable code.

2.1 Purpose

`real_robot.launch.py`:

- Initializes the differential drive controller for wheel-based movement.
- Activates perception for LiDAR and camera data processing.
- Starts navigation for path planning and obstacle avoidance.

`simulation.launch.py`:

- Launches Gazebo for robot and environment simulation.
- Sequences startup with delays to avoid conflicts.
- Runs RViz2 for visualizing sensor and navigation data.

2.2 System Architecture

The system is a differential drive mobile robot with:

- **Hardware (Real)**: Two wheels, LiDAR, cameras, ROS 2 computer.
- **Software**: ROS 2 (Humble+), with `robot_control`, `robot_perception`, `robot_nav`.
- **Simulation**: Gazebo (Fortress+), `robot_model`, RViz2.

Components communicate via ROS 2 topics/services, with modular control, perception, and navigation systems.

2.3 Key Features

- **Dynamic Paths**: Uses `get_package_share_directory` for portability.
- **Modularity**: Employs `IncludeLaunchDescription` for reuse.
- **Timing**: Uses `TimerAction` for simulation sequencing.
- **Visualization**: RViz2 for simulation debugging.
- **Scalability**: Easy to extend with new components.

3 System Components

- `robot_control`: Publishes odometry, subscribes to `/cmd_vel`.
- `robot_perception`: Publishes `/scan`, `/image_raw`.
- `robot_nav`: Runs Nav2 for SLAM and planning.
- `robot_model` (sim): URDF and world files for Gazebo.
- **Gazebo** (sim): Simulates physics.
- **RViz2** (sim): Visualizes `/map`, `/scan`, `/path`.
- **ROS 2 Middleware**: DDS-based communication.

3.1 Component Relationships

The figure below visualizes component dependencies and data flow, with `real_robot.launch.py` (blue) and `simulation.launch.py` (orange) subgraphs. The simulation graph is shifted left for clarity, with colored nodes and annotated arrows.

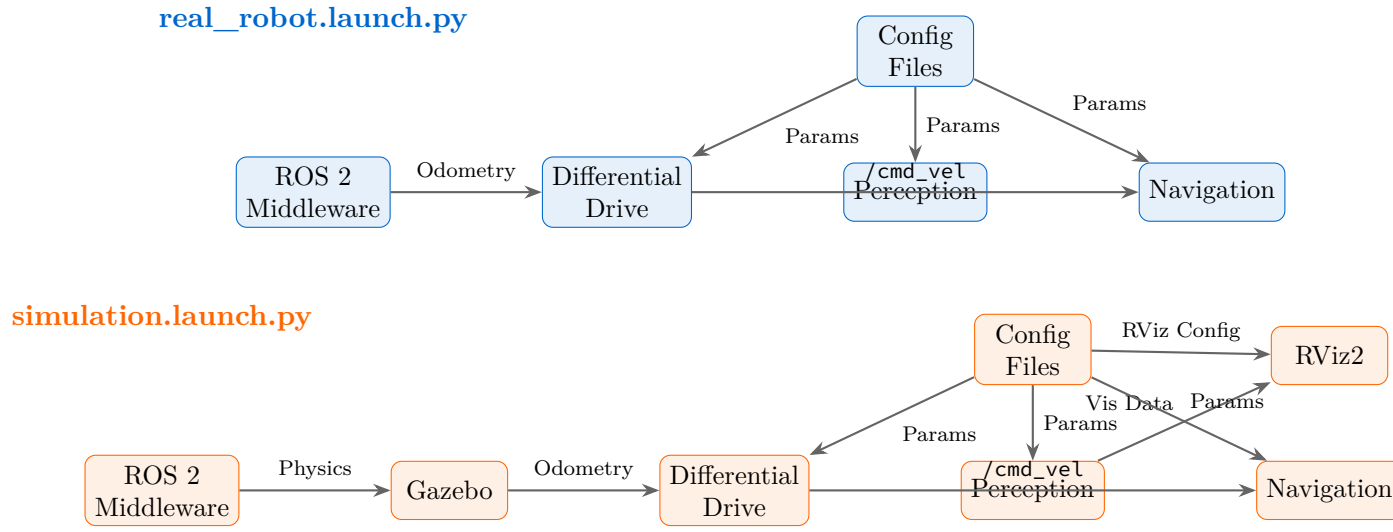

Figure 1: Dependency and Data Flow for Launch Files

4 Launch File Details

4.1 real_robot.launch.py

```python
# real_robot.launch.py
# Launches core components for a physical robot

from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from ament_index_python.packages import get_package_share_directory
import os

def generate_launch_description():
    # Locate package directories
    pkg_robot_control = get_package_share_directory('robot_control')
    pkg_robot_nav = get_package_share_directory('robot_nav')
    pkg_robot_perception = get_package_share_directory('robot_perception')

    # Differential drive controller
    diff_controller_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(pkg_robot_control,
            'launch', 'diff_controller.launch.py'))
    )

    # Perception system
    perception_system_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(pkg_robot_perception,
            'launch', 'perception_system_real.launch.py'))
    )

    # Navigation system
    navigation_system_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(pkg_robot_nav,
            'launch', 'navigation_system.launch.py'))
    )

    return LaunchDescription([
        diff_controller_launch,      # Robot control
        perception_system_launch,    # Sensor processing
        navigation_system_launch     # Navigation
    ])
```

Listing 1: real_robot.launch.py

4.1.1 Purpose

Enables a physical robot's movement, sensing, and navigation for real-world tasks like warehouse automation.

4.1.2 Inputs

- **Packages**: `robot_control`, `robot_perception`, `robot_nav`.
- **Launch Files**: `diff_controller.launch.py`, `perception_system_real.launch.py`, `navigation_system.launch.py`.

4.1.3 Outputs

- **Nodes**: For control, sensors, navigation.
- **Topics**: `/odom`, `/scan`, `/image_raw`, `/path`.
- **State**: Operational robot.

4.1.4 Dependencies

- **Packages**: `robot_control`, `robot_perception`, `robot_nav`.
- **Modules**: `launch`, `ament_index_python`, `os`.
- **Config**: YAML files for parameters.

4.1.5 Configuration

Parameters include wheel base, LiDAR range, map frame. Check included launch files for details.

4.1.6 Error Handling

Errors logged to console. Use `ros2 log` or `rqt_console` for debugging.

4.2 simulation.launch.py

```python
# simulation.launch.py
# Launches a simulated robotic system with Gazebo and RViz2

from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, ExecuteProcess, TimerAction
from launch.launch_description_sources import PythonLaunchDescriptionSource
from ament_index_python.packages import get_package_share_directory
import os

def generate_launch_description():
    # Locate package directories
    pkg_robot_model = get_package_share_directory('robot_model')
    pkg_robot_control = get_package_share_directory('robot_control')
    pkg_robot_nav = get_package_share_directory('robot_nav')
    pkg_robot_perception = get_package_share_directory('robot_perception')

    # Gazebo simulation
    gazebo_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(pkg_robot_model, 'launch', 'gazebo.launch.py'))
    )

    # Differential drive controller with delay
    diff_controller_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(os.path.join(pkg_robot_control, 'launch', 'diff_controller_sim.launch.py'))
    )
    diff_controller_delay = TimerAction(
        period=7.0,  # Wait for Gazebo
        actions=[diff_controller_launch]
    )
```

```
30
31      # Perception system with delay
32      perception_system_launch = IncludeLaunchDescription(
33          PythonLaunchDescriptionSource(os.path.join(pkg_robot_perception,
                'launch', 'perception_system_sim.launch.py'))
34      )
35      perception_delay = TimerAction(
36          period=9.0,  # Wait for controller
37          actions=[perception_system_launch]
38      )
39
40      # Navigation system with delay
41      navigation_system_launch = IncludeLaunchDescription(
42          PythonLaunchDescriptionSource(os.path.join(pkg_robot_nav,
                'launch', 'navigation_system_sim.launch.py'))
43      )
44      navigation_delay = TimerAction(
45          period=12.0,  # Wait for perception
46          actions=[navigation_system_launch]
47      )
48
49      # RViz2 with delay
50      rviz2_node = ExecuteProcess(
51          cmd=['ros2', 'run', 'rviz2', 'rviz2'],
52          output='screen'
53      )
54      rviz_delay = TimerAction(
55          period=15.0,  # Wait for stabilization
56          actions=[rviz2_node]
57      )
58
59      return LaunchDescription([
60          gazebo_launch,        # Simulation
61          diff_controller_delay, # Control
62          perception_delay,     # Sensors
63          navigation_delay,     # Navigation
64          rviz_delay            # Visualization
65      ])
```

Listing 2: simulation.launch.py

4.2.1 Purpose

Sets up a virtual robot for testing, debugging, or algorithm development with Gazebo and RViz2.

4.2.2 Inputs

- **Packages**: `robot_model`, `robot_control`, `robot_perception`, `robot_nav`.
- **Launch Files**: `gazebo.launch.py`, `diff_controller_sim.launch.py`, `perception_system_sim.launch.py`, `navigation_system_sim.launch.py`.
- **Command**: `ros2 run rviz2 rviz2`.

4.2.3 Outputs

- **Nodes**: For simulation, control, perception, navigation, visualization.
- **Topics**: Simulated odometry, sensor, navigation, visualization data.
- **State**: Operational simulated system.

4.2.4 Dependencies

- **Packages**: `robot_model`, `robot_control`, `robot_perception`, `robot_nav`, `rviz2`.
- **Modules**: `launch`, `ament_index_python`, `os`.
- **Tools**: Gazebo, RViz2.
- **Config**: URDF, world, RViz2 files.

4.2.5 Configuration

Includes URDF paths, world files, RViz2 settings. Verify configurations in included files.

4.2.6 Error Handling

Errors logged to console. Use `ros2 topic list` or `rqt_graph` for connectivity checks.

5 Use Cases

- **Warehouse**: `real_robot.launch.py` for goods transport.
- **Development**: `simulation.launch.py` for algorithm testing.
- **Education**: Simulation for ROS 2 learning or research.
- **Debugging**: RViz2 for sensor/navigation issue detection.

6 Best Practices

- **Modularity**: Use smaller, reusable launch files.
- **Parameters**: Load settings via YAML.
- **Logging**: Enable console output, monitor with `rqt_console`.
- **Testing**: Validate in simulation first.
- **Docs**: Comment code, maintain separate docs.

7 Maintenance and Extension

- **Add Components**: Use `IncludeLaunchDescription` or `ExecuteProcess`, adjust delays.
- **Tune Delays**: Modify `TimerAction` periods.
- **Update Packages**: Build and source dependencies.
- **Versioning**: Track changes to avoid regressions.

8 Troubleshooting

- **Package Errors**: Run `colcon build`, source workspace.
- **Gazebo Issues**: Check URDF/world syntax, adjust physics.
- **Topic Issues**: Use `ros2 topic list`, `ros2 node info`.
- **RViz2 Issues**: Verify topic subscriptions, frames.
- **Parameter Issues**: Validate YAML with `ros2 param load`.

9 References

- ROS 2: https://docs.ros.org/en/humble/
- Gazebo: https://gazebosim.org/docs
- RViz2: https://github.com/ros-visualization/rviz
- Launch System: https://docs.ros.org/en/humble/Tutorials/Launch/Launch-system.html

10 Conclusion

This modern guide details `real_robot.launch.py` and `simulation.launch.py`, offering a vibrant, clear resource for ROS 2 developers. With a colorful dependency graph, detailed code breakdowns, and practical advice, it empowers efficient development and maintenance of robotic systems for real-world and simulated environments.