Perception System Launch Files Documentation

xAI Generated Documentation

May 13, 2025

Contents

1 Introduction

This document provides a detailed technical overview of the launch files for a ROS 2-based perception system designed for robotic applications. The system focuses on initializing hardware and software components to enable hand detection and other perception tasks. The two key files analyzed are `Perception_system_real.launch.py`, which configures the system for real hardware (RPLidar and Raspberry Pi camera), and `Perception_system_sim.launch.py`, which sets up a simulated environment using a USB camera and visualization tools. This 10-page documentation covers the purpose, structure, components, and workflow of these files, providing code snippets, diagrams, and insights into their roles within the broader perception system. It is intended for developers, engineers, and roboticists working with ROS 2.

2 System Overview

The perception system is designed to enable robots to detect human interactions, such as raised hands, using camera and LiDAR data. The launch files serve as the entry point, orchestrating the startup of ROS 2 nodes for hardware drivers, perception processing, and visualization. The system supports two environments:

- **Real Setup**: Uses RPLidar and Raspberry Pi camera for real-world deployment.
- **Simulated Setup**: Uses a USB camera and RQT visualization for testing and development.

Both launch files initialize nodes that publish sensor data to topics like `/image_raw` and `/scan`, which are consumed by perception nodes for processing. The launch files also reference a configuration file (`detector_params.yaml`) for parameter tuning.

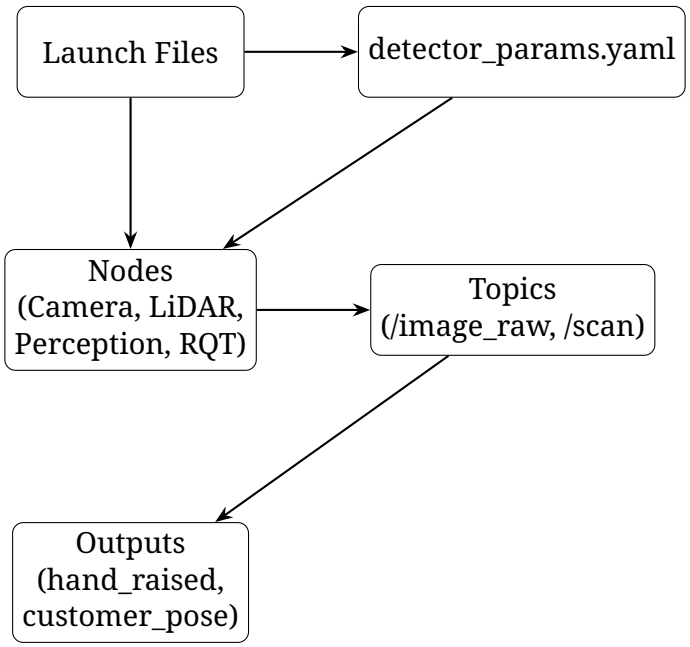

Figure 1: System Architecture

3 File Descriptions

3.1 Perception_system_real.launch.py

The `Perception_system_real.launch.py` file is a Python-based ROS 2 launch script that initializes the perception system for a real robot equipped with an RPLidar sensor and a Raspberry Pi camera. It is responsible for starting hardware drivers and perception nodes, ensuring seamless integration with the robot's physical environment.

3.1.1 Purpose and Functionality

- **Purpose**: Configures and launches nodes to interface with real hardware, enabling data acquisition and perception processing.
- **Functionality**: Initializes RPLidar for depth sensing, Raspberry Pi camera for imaging, and perception nodes for hand detection and emotion recognition.
- **Output**: Publishes sensor data to `/scan` (LiDAR) and `/image_raw` (camera), consumed by downstream nodes.

3.1.2 Components and Structure

The file follows a modular structure:

- **Imports**: Includes ROS 2 launch utilities (`launch`, `launch_ros.actions`), package directory access (`ament_index_python.packages`), and `os` for file path manipulation.
- **Function**: `generate_launch_description()` constructs a `LaunchDescription` object containing node definitions.
- **Nodes**:
  - **RPLidar Node**: Configures the RPLidar sensor for depth data.
  - **Camera Node**: Interfaces with the Raspberry Pi camera for image streaming.
  - **Perception Node**: Executes `perception.py` for hand detection and pose estimation.
  - **Emotion Recognition Node**: Runs `Emotion_Recognition.py` for additional perception tasks.

Listing 1: Core Structure of Perception_system_real.launch.py

```
#!/usr/bin/env python3
from launch import LaunchDescription
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory
import os

def generate_launch_description():
    pkg_dir = get_package_share_directory('robot_perception')
    pkg_rplidar_ros = get_package_share_directory('rplidar_ros')
    params_file = os.path.join(pkg_dir, 'config', 'detector_params.yaml')
    # Node definitions follow
```

3.1.3 Key Component: RPLidar Node

The RPLidar node is critical for providing depth data, enabling 3D pose estimation.

- **Package**: `rplidar_ros`.
- **Executable**: `rplidar_composition`.
- **Parameters**:
  - `serial_port`: `/dev/ttyUSB1` for hardware connection.
  - `serial_baudrate`: 115200 (for A1/A2 models, with a commented option for 256000 for A3).
  - `frame_id`: `lidar_link` for TF2 integration.
  - `inverted`: False, `angle_compensate`: True for accurate scans.
- **Output**: Publishes `sensor_msgs/LaserScan` messages to `/scan`.

Listing 2: RPLidar Node Configuration

```
rplidar_node = Node(
    name='rplidar_composition',
    package='rplidar_ros',
    executable='rplidar_composition',
    output='screen',
    parameters=[{
        'serial_port': '/dev/ttyUSB1',
        'serial_baudrate': 115200,
        'frame_id': 'lidar_link',
        'inverted': False,
        'angle_compensate': True,
    }],
)
```

3.1.4 Key Component: Camera Node

The camera node streams images from the Raspberry Pi camera using the `v4l2_camera` package.

- **Package**: `v4l2_camera`.
- **Executable**: `v4l2_camera_node`.
- **Parameters**:
  - `video_device`: `/dev/video0`.
  - `image_size`: [640, 480] pixels.
  - `camera_frame_id`: `camera_link` for TF2.
- **Output**: Publishes `sensor_msgs/Image` messages to `/image_raw`.

Listing 3: Camera Node Configuration

```
camera_node = Node(
    package='v4l2_camera',
    executable='v4l2_camera_node',
    name='camera_node',
    parameters=[{
```

```
        'video_device': '/dev/video0',
        'image_size': [640, 480],
        'camera_frame_id': 'camera_link'
    }]
)
```

3.1.5 Key Component: Perception and Emotion Recognition Nodes

- **Perception Node**: Executes `perception.py` from the `robot_perception` package, processing sensor data for hand detection and pose estimation.
- **Emotion Recognition Node**: Runs `Emotion_Recognition.py`, likely processing `/image_raw` for facial or emotional analysis.
- **Output**: Both nodes produce outputs (e.g., `/hand_raised`, `/customer_pose`) for downstream use.

Listing 4: Perception Node Configuration

```
perception_node = Node(
    package='robot_perception',
    executable='perception.py',
    output='screen'
)
```

3.1.6 Flow Within the File

The internal flow of `Perception_system_real.launch.py` is as follows:

1. Retrieve package directories for `robot_perception` and `rplidar_ros`.
2. Construct the path to `detector_params.yaml` for parameter loading.
3. Define nodes for RPLidar, camera, perception, and emotion recognition.
4. Return a `LaunchDescription` object to launch all nodes.

3.2 Perception_system_sim.launch.py

The `Perception_system_sim.launch.py` file is a ROS 2 launch script designed for a simulated environment, replacing the RPLidar and Raspberry Pi camera with a USB camera and adding a visualization tool for debugging.

3.2.1 Purpose and Functionality

- **Purpose**: Facilitates testing and development by simulating the perception system without physical hardware.
- **Functionality**: Initializes a USB camera for imaging, perception nodes for processing, and RQT for visualizing debug outputs.
- **Output**: Publishes `/image_raw` and debug images to `/hand_detection_debug`.

3.2.2 Components and Structure

- **Imports**: Includes ROS 2 launch utilities, `ament_index_python.packages` for package paths, and `os`.
- **Function**: `generate_launch_description()` defines the launch configuration.
- **Nodes**:
  - **USB Camera Node**: Streams simulated camera data.
  - **Perception Node**: Executes `perception.py`.
  - **Emotion Recognition Node**: Runs `Emotion_Recognition.py`.
  - **RQT Image View Node**: Visualizes debug images.

Listing 5: Core Structure of Perception_system_sim.launch.py

```
#!/usr/bin/env python3
from launch import LaunchDescription
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory
import os

def generate_launch_description():
    pkg_dir = get_package_share_directory('robot_perception')
    params_file = os.path.join(pkg_dir, 'config', 'detector_params.yaml')
    # Node definitions follow
```

3.2.3 Key Component: USB Camera Node

The USB camera node simulates the camera input for testing.

- **Package**: `usb_cam`.
- **Executable**: `usb_cam_node_exe`.
- **Parameters**:
  - `video_device`: `/dev/video0`.
  - `image_width`, `image_height`: 640x480.
  - `pixel_format`: yuyv.
  - `framerate`: 30.0.
  - `camera_frame_id`: `camera_link`.
- **Remappings**: Maps `image` to `/image_raw`.
- **Output**: Publishes `sensor_msgs/Image` to `/image_raw`.

Listing 6: USB Camera Node Configuration

```
camera_node = Node(
    package='usb_cam',
    executable='usb_cam_node_exe',
    name='usb_cam',
    parameters=[{
        'video_device': '/dev/video0',
        'image_width': 640,
```

```
        'image_height': 480,
        'pixel_format': 'yuyv',
        'camera_frame_id': 'camera_link',
        'framerate': 30.0,
    }],
    remappings=[
        ('image', '/image_raw'),
    ]
)
```

3.2.4 Key Component: RQT Image View Node

The RQT node provides real-time visualization of debug images.

- **Package**: `rqt_image_view`.
- **Executable**: `rqt_image_view`.
- **Arguments**: Subscribes to `/hand_detection_debug`.
- **Output**: Displays annotated images for debugging.

Listing 7: RQT Image View Node Configuration

```
rqt_image_view = Node(
    package='rqt_image_view',
    executable='rqt_image_view',
    name='rqt_image_view',
    arguments=['/hand_detection_debug']
)
```

3.2.5 Key Component: Perception and Emotion Recognition Nodes

These nodes are identical to those in the real setup, ensuring consistency in processing logic across environments.

3.2.6 Flow Within the File

The internal flow of `Perception_system_sim.launch.py` is:

1. Retrieve the `robot_perception` package directory.
2. Construct the path to `detector_params.yaml`.
3. Define nodes for USB camera, perception, emotion recognition, and RQT.
4. Return a `LaunchDescription` object to launch all nodes.

4 System Workflow

The perception system's workflow, driven by the launch files, is as follows:

1. **Launch Initialization**:
    - `Perception_system_real.launch.py` starts:
        - RPLidar node, publishing `/scan`.

  - Raspberry Pi camera node, publishing `/image_raw`.
  - Perception and emotion recognition nodes.
- `Perception_system_sim.launch.py` starts:
  - USB camera node, publishing `/image_raw`.
  - Perception and emotion recognition nodes.
  - RQT node for visualizing `/hand_detection_debug`.

2. **Parameter Loading**: Both launch files reference `detector_params.yaml` to configure perception nodes.
3. **Data Acquisition**:
   - In the real setup, RPLidar and camera nodes provide depth and image data.
   - In simulation, the USB camera provides image data.
4. **Data Processing**:
   - The perception node (`perception.py`) subscribes to `/image_raw` and `/scan` (real setup only), processing data to detect hands and estimate poses.
   - The emotion recognition node likely processes `/image_raw` for additional tasks.
5. **Output Publication**:
   - The perception node publishes:
     - `/hand_raised`: Boolean indicating hand status.
     - `/customer_pose`: 3D pose of the detected person.
     - `/hand_detection_debug`: Annotated images for debugging.
   - The RQT node (simulation only) visualizes debug images.
6. **Downstream Integration**: Outputs are consumed by other robotic components (e.g., navigation or interaction modules).

5 Relationships Between Files

The two launch files are closely related, serving as entry points for different environments:

- **Shared Components**:
  - Both launch `perception.py` and `Emotion_Recognition.py` from the `robot_perception` package, ensuring consistent processing logic.
  - Both reference `detector_params.yaml` for parameter configuration.
- **Differences**:
  - `Perception_system_real.launch.py` includes RPLidar for depth data and uses `v4l2_camera` for the Raspberry Pi camera.
  - `Perception_system_sim.launch.py` uses `usb_cam` for simulation and adds `rqt_image_view` for visualization.

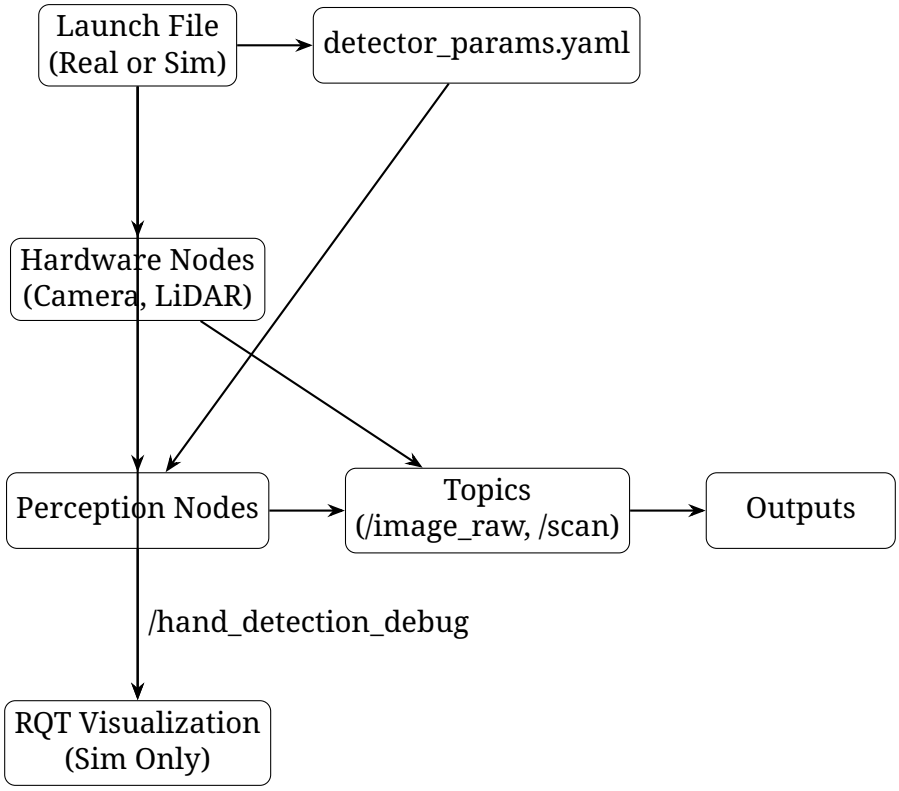

Figure 2: System Workflow Diagram

- **Data Flow**:
  - Both files initialize nodes that publish to `/image_raw`, consumed by perception nodes.
  - The real setup additionally publishes `/scan` for depth processing.
  - Outputs from perception nodes are consistent across setups, enabling seamless transitions between real and simulated environments.

6 Deployment Considerations

To deploy the launch files:

- **Real Setup**:
  - Ensure RPLidar and Raspberry Pi camera are connected to `/dev/ttyUSB1` and `/dev/video0`, respectively.
  - Install dependencies: `rplidar_ros`, `v4l2_camera`, `robot_perception`.
  - Run: `ros2 launch robot_perception Perception_system_real.launch.py`.
- **Simulation Setup**:
  - Connect a USB camera to `/dev/video0`.
  - Install dependencies: `usb_cam`, `rqt_image_view`, `robot_perception`.
  - Run: `ros2 launch robot_perception Perception_system_sim.launch.py`.

- **Configuration**: Verify `detector_params.yaml` exists in the `robot_perception/config` directory.

7 Conclusion

The `Perception_system_real.launch.py` and `Perception_system_sim.launch.py` files are critical components of a ROS 2-based perception system, enabling flexible deployment in real and simulated environments. By initializing hardware drivers, perception nodes, and visualization tools, they ensure robust data acquisition and processing for hand detection and related tasks. This documentation provides a comprehensive understanding of their structure, components, and workflow, serving as a valuable resource for roboticists developing perception-enabled systems.