# ROS 2 Advanced Programming & Simulation - Complete Recap & Cheat Sheet

## Training Overview

Goal: Develop advanced ROS2 skills with emphasis on modularity, simulation, and communication between robotic systems.

## Module 2.1: ROS2 Packages & Launch Architecture

### What We Accomplished

- Mastered scalable ROS2 project structures and multi-package management
- Automated complex multi-node systems using advanced launch files
- Implemented dynamic configurations with launch arguments and substitutions
- Created reactive systems using event handlers for process monitoring

### Key Takeaways for Students

- Launch files replace manual node startup with single-command automation
- Dynamic launch arguments make systems flexible and reusable
- Event handlers enable reactive architectures that respond to node states
- Multi-package integration allows building complex systems from modular components
- Proper package structure ensures maintainability and team collaboration

### Essential Commands

```
# Build specific packages from our examples

colcon build --packages-select simple_service my_py_pkg



# Launch basic calculator system
```

```bash
ros2 launch simple_service basic_calculator.launch.py

# Launch with dynamic arguments (like we did with calculator)
ros2 launch simple_service dynamic_calculator.launch.py a:=25 b:=15
operation:=multiply

# Show available launch arguments
ros2 launch simple_service dynamic_calculator.launch.py --show-args

# Launch multi-package system (nodes from different packages)
ros2 launch simple_service multi_package_system.launch.py

# Analyze node connections
rqt_graph
```

# Module 2.2: Simulation in Gazebo Fortress

## What We Accomplished

- Created and controlled simulated robots using Gazebo + ROS2 integration
- Mastered URDF/Xacro for comprehensive robot modeling
- Configured and visualized robot behavior in RViz2
- Implemented robot control through ROS2 topics and services
- Built complete simulation workflows from modeling to visualization

## Key Takeaways for Students

- Simulation enables risk-free development before hardware deployment
- URDF defines robot structure (links, joints, sensors, inertial properties)
- Xacro extends URDF with macros and parameters for reusable components
- Gazebo provides physics simulation while RViz2 focuses on data visualization
- ROS2 Control manages communication between algorithms and simulation
- Standard topics (/cmd_vel, /odom, /scan) enable universal robot control

## Essential Commands

```bash
# Install TurtleBot3 simulation packages (like we did)

sudo apt install ros-humble-turtlebot3*



# Set TurtleBot3 model environment variable

export TURTLEBOT3_MODEL=burger



# Launch TurtleBot3 in Gazebo world

ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

```
# Control robot with keyboard (as practiced)

ros2 run turtlebot3_teleop teleop_keyboard



# Launch RViz2 with TurtleBot3 configuration

ros2 run rviz2 rviz2 -d $(ros2 pkg prefix
turtlebot3_bringup)/share/turtlebot3_bringup/rviz/turtlebot3.rviz



# Monitor robot topics (like we analyzed)

ros2 topic list

ros2 topic echo /odom

ros2 topic echo /cmd_vel

ros2 topic echo /scan



# Check TF frames

ros2 topic echo /tf
```

# Module 2.3: ROS2 Communication and Middleware

## What We Accomplished

- Deepened understanding of DDS (Data Distribution Service) as ROS2's foundation
- Mastered QoS (Quality of Service) policies for different communication scenarios
- Implemented real factory communication system with mixed QoS requirements
- Analyzed performance trade-offs between reliability and speed
- Built comprehensive monitoring and analysis tools

## Key Takeaways for Students

- DDS provides decentralized communication - nodes discover each other automatically
- QoS policies control data delivery behavior, not just the data itself
- RELIABLE QoS ensures message delivery for critical systems (safety commands)
- BEST_EFFORT QoS maximizes performance for high-frequency data (sensors)
- Different scenarios require different QoS profiles - no one-size-fits-all
- Services inherently use RELIABLE QoS for request-response patterns
- Mixed QoS systems allow optimizing different parts of your application

## Essential Commands

```bash
# Launch complete factory communication system (from our project)

ros2 launch factory_communication factory_demo.launch.py



# Monitor topic rates (like we analyzed performance)

ros2 topic hz /factory/emergency
```

```bash
ros2 topic hz /factory/sensors/temperature

# Check QoS settings of topics
ros2 topic info /factory/emergency --verbose
ros2 topic info /factory/sensors/temperature --verbose

# Call calculator service (as we tested)
ros2 service call /factory/quality_calculator
example_interfaces/srv/AddTwoInts "{a: 5, b: 3}"

# Monitor all active nodes
ros2 node list

# Check node info and connections
ros2 node info /emergency_controller

# Bag recording for analysis (like we did for sensor data)
ros2 bag record /odom /scan /cmd_vel
```

## Integration Patterns Mastered

### System Architecture Principles

- Modular Design: Break complex systems into manageable packages
- Configuration Management: Use launch files for reproducible setups
- Communication Strategy: Match QoS policies to data criticality
- Visualization Integration: Combine Gazebo physics with RViz2 data display

### Development Workflow

1. Model → Define robot structure with URDF/Xacro
2. Simulate → Test behavior in Gazebo with physics
3. Visualize → Monitor data streams in RViz2
4. Control → Implement algorithms using ROS2 topics/services
5. Optimize → Apply appropriate QoS for performance requirements

## Core Competencies Gained

### Technical Skills

- Advanced ROS2 package management and launch file creation
- Robot modeling with URDF/Xacro and simulation in Gazebo
- ROS2 communication middleware understanding and QoS implementation
- Multi-node system integration and event-driven architectures

### Conceptual Understanding

- Trade-offs between simulation and real hardware
- Communication reliability vs. performance considerations
- Modular system design principles
- Real-time data distribution patterns

## Practical Applications

- Factory automation systems with mixed criticality requirements
- Robot navigation and control systems
- Sensor data processing pipelines

## Progress Assessment

Students completing this training can now:

- Design and implement complex multi-package ROS2 systems
- Create and control simulated robots in Gazebo environments
- Apply appropriate communication strategies using QoS policies
- Build integrated simulation-visualization-control pipelines
- Develop production-ready robotic applications with proper architecture

This foundation enables tackling real-world robotics challenges with professional-grade ROS2 development practices.