ROS 2 SLAM Resource Management using Reinforcement Learning

Analysis under CPU Contention in Simulation

Group 19

Kieu Son Tung Nguyen Phu An Nguyen Thanh Lam Pham Chi Bang

Hanoi University of Science and Technology (HUST)
Class: DSAI-02-K68

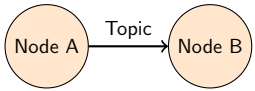
February 3, 2026

Outline

1. Introduction to ROS 2

Robot Operating System 2 (ROS 2) is the industry standard middleware for robotics.

- **Nodes:** Independent processes performing specific tasks (e.g., Sensing, Planning, Actuation).
- **Topics:** Publish/Subscribe mechanism for data exchange.
- **SLAM Toolbox:** A popular package for 2D Simultaneous Localization and Mapping.



Key Feature

ROS 2 uses DDS (Data Distribution Service) for real-time communication.

2. Problem Statement: Resource Contention

The Challenge:

- ROS 2 nodes operate as independent OS processes.
- On hardware with limited resources (CPU/RAM), nodes compete for execution time.

2. Problem Statement: Resource Contention

The Challenge:

- ROS 2 nodes operate as independent OS processes.
- On hardware with limited resources (CPU/RAM), nodes compete for execution time.

The Symptom:

Without Management

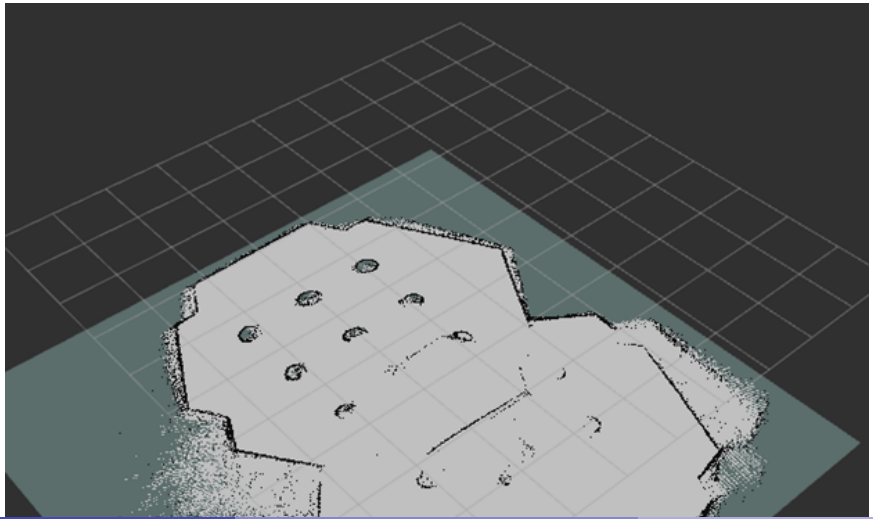
When background tasks (image processing, data logging) spike:

1. **Starvation:** Critical SLAM nodes lose CPU cycles.
2. **Latency:** Topic messages (`/scan`) are delayed.
3. **Failure:** Map drift, ghost artifacts, and navigation failure.

$\rightarrow$ **Need for an Intelligent Resource Manager.**

Visualizing the Problem: Map Drift

Effect of CPU Starvation on Mapping



3. Experimental Setup

Environment:

- **Robot:** TurtleBot3 (Burger) in Gazebo Simulation.
- **Stress Injector:** `cpu_hog` nodes (consume 60-100% CPU).
- **Metrics:** Jitter (σ_t), Throughput (Hz), Map Quality.

4. Reinforcement Learning Approach

We treat Resource Management as a decision-making problem.

Multi-Armed Bandit (MAB)

- *Stateless*: Does not consider current CPU load.
- Algorithm: ϵ-greedy.
- Action: Fix Priority.
- *Pro:* Simple. *Con:* Ignores context.

Q-Learning (MDP)

- *State-Aware*: State $S \in \{Safe, Warning, Critical\}$.
- Update Rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s', a') - Q(s,a)]$$

- *Pro:* Anticipatory behavior.

Action Space: {Normal Priority, High Priority SLAM, Throttle Hog}
Reward: $+1$ if Jitter $< 50ms$, else -1.

5. Experimental Results

Scenario: High Contention (2 Hog Processes active).

Method	Scan Rate (Hz)	Jitter (ms)	Map Quality
Baseline (None)	1 ± 0.8	120	Poor (Drift)
Rule-Based	1.1 ± 0.5	75	Medium
Bandit (MAB)	1.3 ± 0.2	48	Medium-High
Q-Learning	$\mathbf{1.5 \pm 0.2}$	**25**	**High (Stable)**

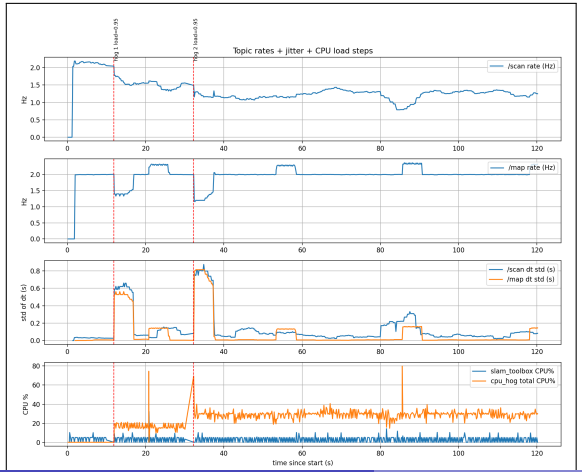
Table: Performance metrics comparison over 120s episode.

- **Baseline:** SLAM starves, leading to broken maps.
- **Q-Learning:** Automatically preempts "Hog" processes when Jitter rises, maintaining 4.8Hz stability.

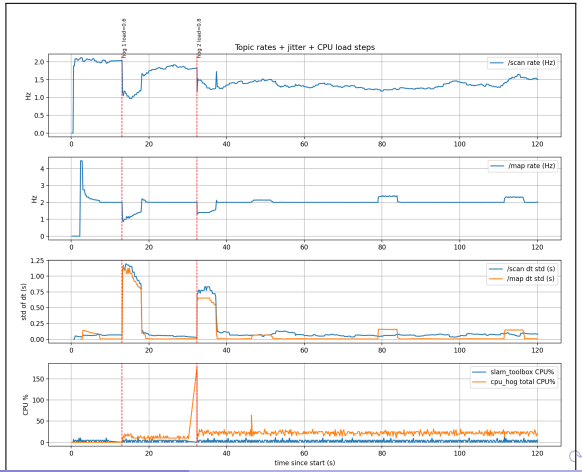
Performance Comparison: Bandit vs. Q-Learning

Learning Convergence & Stability

1. Bandit (ϵ-greedy)



2. Q-Learning (State-Aware)



6. Limitations

Despite positive results, several factors impact the fidelity:

1. **WSL2 Virtualization:**
   - Running on Windows Subsystem for Linux (WSL2) introduces scheduler noise from the Windows host.
   - Not a true "Real-Time" kernel (RT-Preempt).
2. **Resource Constraints:**
   - Limited to virtual cores; difficult to isolate CPU affinity strictly compared to bare-metal Linux.
3. **System Complexity:**
   - ROS 2 Middleware (DDS) has its own internal buffering which RL does not directly control.

Thank you for listening!