UV Dosage Based Coverage Algorithm

POC Document

Version 1.0

CONTENTS:

1 Document Details 3

2 Project Overview 3

3 Problem Statement 3

4 Proof of Concept (POC) 4

4.1 Objectives of the POC 4

4.2 Scope of the POC 4

4.3 Key Features/Functionality to be Demonstrated 5

4.4 Technology Stack (for POC) 5

4.5 Success Criteria for the POC 6

4.6 Dosage Level Classification and User Input Requirement 6

5 Methodology and Approach 7

6 Deliverable 7

7 Estimated Timeline and Resources 7

8 Risks and Mitigation 8

9 Conclusion and Next Steps 8

1 Document Details

Document Title: UV Dosage based Coverage algorithm tracker using ROS

Version: 1.0

Date: June 25, 2025

Author(s): Divagar.N, Nishanth.S, Madhavan.S

2 Project Overview

Software Name: UV Dosage based Coverage algorithm tracker using ROS (ROS Node: uv_dose_tracker)

Brief Description: This ROS - based system tracks and visualizes a robot's trail on a global costmap, simulating the accumulation of dosage (e.g., for UV disinfection, radiation, or any exposure-based activity) as the robot moves. It operates by reading the robot's position, defining a circular footprint, receiving UV dosage level as input from the algorithm, accumulating the dosage within that footprint, assigning trail colors based on dosage levels, drawing the updated trail on a copy of the occupancy grid, and publishing the modified map.

Target Audience/Users: Robotics developers, researchers, industrial users deploying robots for disinfection, radiation mapping, or other exposure-based applications.

Overall Goal/Vision: To develop a robust and reliable ROS-based system for real-time tracking and visualization of robot trails and simulated dosage accumulation on a map dynamically, providing valuable data for coverage assessment and operational insights.

3 Problem Statement

The Problem: There is a need for a system to accurately track and visually represent a robot's path and accumulated "dosage" (e.g., UV exposure, radiation) on a map. Without such a system, it's difficult to verify coverage, identify underexposed areas, or understand the history of a robot's presence in a given environment during operations like disinfection.

Limitations:

- If the goal lies outside the occupancy grid, the algorithm will continuously retry reaching the goal, resulting in a long wait time and failure to progress.

4 Proof of Concept (POC)

4.1 Objectives of the POC

- To develop a working ROS - based system that tracks and visualizes a robot's trail on a map.
- To simulate dosage accumulation based on robot presence and movement over time.
- To assign visual trail colors on the map according to accumulated dosage levels.
- To demonstrate the ability to publish a modified occupancy grid (/map_modified) for visualization (e.g., in Rviz).
- To ensure the system can log dosage values and thresholds on launch.
- To implement threading for separate TF processing and map publishing.

4.2 Scope of the POC

- **In Scope:**
  - Development of the map_trail_persistence.py ROS node.
  - Input handling for /move_base/global_costmap/costmap and TF (for map → base_link transform).
  - Process loop running at a fixed rate (1 Hz) including:
    - Reading robot's position using TF.
    - Defining a circular footprint based on robot_radius.
    - Accumulating dosage value proportional to time for cells within the footprint.
    - Assigning trail colors (TRAIL_COLOR1 to TRAIL_COLOR4) based on dosage levels and disinfect_type threshold.
    - Drawing the updated trail on a copy of the occupancy grid.
    - Publishing the new map to /map_modified.
  - Implementation of adjustable parameters: disinfect_type, dosage_value, robot_radius.
  - Logging of dosage_value and threshold on launch.
  - Use of threading to separate TF processing and map publishing.

- **Out of Scope:**
    - Advanced visualization features beyond publishing the modified map to /map_modified (e.g., built-in RViz features or custom report UIs).
    - Detailed logging of dose statistics.
    - Saving coverage maps to persistent storage.
    - Development of a report UI.
    - Creation of a launch file for running everything cleanly (this POC focuses on the node itself).
    - Integration with other robotic functionalities (e.g., path planning directly influenced by dosage map).

4.3 Key Features/Functionality to be Demonstrated

Real-time Robot Position Tracking: The system will demonstrate its ability to accurately read the robot's current position from TF.

Dynamic Footprint Calculation: Visualization of a circular footprint around the robot, dynamically calculating affected map cells.

Dosage Accumulation: Demonstration of how dosage values increase in visited map cells over time.

Trail Color Mapping: Visual representation of different dosage levels on the map using distinct "trail color codes" (negative grid values: -2, -50, -100, -150).

Map Modification and Publication: The system will publish a continuously updated map_modified topic, showing the robot's trail and accumulated dosage.

Parameter Adjustability: Verification that disinfect_type, dosage_value, and robot_radius can be configured to influence the simulation.

4.4 Technology Stack (for POC)

Category	Details
Core Framework	Robot Operating System (ROS)
Programming Language	Python
ROS Components	rospy, tf (tf2), nav_msgs.msg.OccupancyGrid
Other Tools/Libraries	Standard Python libraries for data structures and calculations
Deployment Environment	ROS-compatible environment (e.g., Ubuntu

	with ROS installed)

4.5 **Success Criteria for the POC**

- The map_trail_persistence.py ROS node launches and runs without critical errors.
- The node successfully subscribes to /move_base/global_costmap/costmap and listens to TF transforms (map → base_link).
- The /map_modified topic is published continuously at approximately 1 Hz.
- The published /map_modified visually displays a trail behind the robot, with colors corresponding to accumulated dosage levels as defined in the "Trail Color Codes" table.
- Adjustable parameters (disinfect_type, dosage_value, robot_radius) can be set on launch and correctly influence the dosage simulation and trail coloring.
- Dosage value and threshold are logged to the console upon node launch.
- The system demonstrates threading for separate TF processing and map publishing, ensuring smooth operation.
- The system can be successfully tested using rosrun your_package map_trail_persistence.py when /move_base/global_costmap/costmap and TF are active.

4.6 **Dosage Level Classification and User Input Requirement**

To enable effective visualization and simulation of UV dosage accumulation, four distinct dosage levels are defined, each represented by a specific color.

Level	Color	Dosage Threshold Description
Level 1	Yellow	**Level 1** corresponds to **index 0** → Threshold value: **10**
Level 2	Orange	**Level 2** corresponds to **index 1** → Threshold value: **20**
Level 3	Red	**Level 3** corresponds to **index 2** → Threshold value: 3**0**
Level 4	Green	**Level 4** corresponds to **index 3** → Threshold value: 4**0**

Customization: Users can modify these threshold values directly in the **dose_calculator.py** class file according to the project or use-case requirements. Each level's behavior can be adjusted by updating the corresponding index in the threshold_map

For effective tracking and simulation, the user must configure dosage threshold values for all four levels (Level 1 to Level 4). These threshold values will be used to categorize accumulated dosage on the map and assign the corresponding trail color. Proper configuration ensures accurate dosage visualization and operational insights during disinfection or exposure-based activities.

5 Methodology and Approach

Development Approach: Iterative development focusing on getting core functionalities (input, processing loop, output) working first, then refining with parameters and threading.

Testing Strategy: Manual testing within a ROS simulation environment (e.g., Gazebo with a robot model publishing TF and a costmap) and visual verification in Rviz. Unit tests for core logic components will be considered if time permits.

Feedback Collection: Internal review with stakeholders and subject matter experts to validate simulation logic and visual accuracy.

Key Milestones:

- **Week 1:** Initial setup of ROS environment and basic map_trail_persistence.py node structure. Successful subscription to /move_base/global_costmap/costmap and TF data.
- **Week 2:** Implementation of robot footprint calculation, initial dosage accumulation logic, and basic map modification. First successful publication of /map_modified with a simple trail.
- **Week 3:** Integration of trail coloring logic based on dosage thresholds. Implementation of adjustable parameters. Refinement of map updates and performance.
- **Week 4:** Addition of threading for TF processing and publishing. Final testing, documentation of usage, and preparation for POC demonstration.

6 Deliverable

- Working **map_trail_persistence.py** ROS node.
- **Source code** for map_trail_persistence.py.
- A brief **README/documentation** explaining how to run and test the node, including parameters.
- A demonstration of the running POC in a ROS environment (e.g., **video**).
- Summary report of POC findings against success criteria.

7 Estimated Timeline and Resources

Estimated Start Date: [Current Date]

Estimated Completion Date: [Approx. 4 weeks from Start Date]

Team Members & Roles:

- Divagar.N : ROS Developer
- Madhavan.S : ROS Developer
- Nishanth.S : ROS Developer

Required Resources:

- A computer with a Linux OS (Ubuntu: 20.04LTS)
- ROS installation (Noetic)
- ROS packages for navigation stack (for costmap and TF)
- Rviz for visualization

8 Risks and Mitigation

Risk	Mitigation Strategy
TF transform delays/errors	Implement robust error handling for TF lookups; potentially use TF buffer for older transforms if real-time is critical but unreliable.
Performance issues with map processing	Optimize map iteration and dosage calculation; consider using NumPy for efficient array operations if not already.
ROS environment setup complexities	Provide clear, step-by-step setup instructions; utilize Docker or pre-configured VMs for consistency.
Inaccurate dosage simulation	Rigorous testing with various disinfect_type and dosage_value settings; involve domain experts for validation.
Visual representation clarity in Rviz	Experiment with different trail color mappings and grid values to ensure clear visual differentiation.

9 Conclusion and Next Steps

Conclusion: This POC aims to successfully demonstrate the core functionality of tracking robot trails, simulating dosage accumulation, and visualizing it on a ROS

map. A successful POC will validate the feasibility of the concept and provide a strong foundation for future development stages.

Potential Next Steps (Post - POC):

- **Stage 1:** Evaluate POC results against the defined success criteria.
- **Stage 2:** Develop advanced visualization features (e.g., improved Rviz plugins, logging detailed dose statistics for analysis).
- **Stage 3:** Implement functionalities for saving coverage maps persistently and/or creating a dedicated report UI for in-depth analysis.
- Develop a comprehensive ROS launch file for easy deployment and configuration of all system components.
- Consider integrating the dosage map with robot navigation for coverage path planning.