Project Proposal

Project tile: Autonomous Hazardous Waste Foraging System (Codename: Swarm)
Category: Collection robots (>10 robots)
Group No: 5
***Team members*:** Aakash Shetty Dammala (asd@umd.edu) and Dayanidhi Kandade (dotsv@umd.edu)

Overview and Business Case

When disasters like Japan's nuclear leak happen, cleaning up radioactive waste becomes incredibly dangerous for human workers. We see an opportunity for Acme Robotics to enter the hazardous materials sector with autonomous systems that keep people out of harm's way.

We're proposing Project Swarm - a decentralized robot team that can autonomously find, collect, and contain hazardous waste (we'll simulate it with neon green objects). By deploying more than 10 TurtleBot3 robots working together, we can quickly cover large contaminated areas. The key challenge we're tackling: the robots won't know where objects are ahead of time. Locations will be randomized each run, just like in real disaster scenarios. With Acme's recent venture funding, now's the perfect time to position ourselves as leaders in disaster response robotics.

Technical Architecture

We're building on proven hardware: TurtleBot3 Waffle Pi robots running in Webots simulation with ROS 2 Humble. Each robot carries a 2D LiDAR for avoiding obstacles, an RGB camera for spotting waste, and odometry for tracking position. On the software side, we're using C++14/17 with OpenCV for computer vision (HSV color filtering and blob detection) and Nav2 for navigation. Everything will be Apache 2.0 licensed to keep it commercially viable.

The system is decentralized - no central command. Each robot loads a YAML config file that tells it which zone to clean and where to dump collected items, then operates independently.

Our robots follow a ***Finite State Machine with five behaviors***: First, they navigate to their assigned cleanup zone using Nav2. Then they search the area while scanning with their camera by performing in-place turn. When they detect something neon green, they use visual servoing to line up with it. Next, they push the object toward the dump point while continuously correcting their heading. Finally, they back up to release the object. If they hit a wall or obstacle closer than 0.2m at any point, they automatically reverse and turn to recover.

Key assumptions: Objects stay put once placed, they're randomly distributed each run, the arena has clear boundaries, robots only talk via ROS 2 topics (no direct coordination), and the simulation physics behaves predictably.

Development Process

We're using a 2-week Agile sprint. Week 1 focuses on setting up the robots, implementing the FSM, building the perception system with OpenCV, and creating the Webots environment. Week 2 is all about scaling to 10+ robots, integrating Nav2 navigation, and fine-tuning the robot’s behavior.
We'll work in pairs throughout - one person driving the code while the other reviews logic, maintains UML diagrams, and manages the GitHub repo. We'll swap roles each sprint to keep things fresh.

For quality assurance, we're writing unit tests with GTest, running integration tests that scale from 1 to N robots, documenting everything with Doxygen, setting up continuous integration with GitHub Actions, and doing regular code reviews during our pair programming sessions.

Open-Source Resources

We're building on solid foundations: ROS 2 Humble, OpenCV (BSD), Nav2 (Apache 2.0), Webots (Apache 2.0), GTest, and Doxygen. We're also looking at REMROC (MIT license) and Nav2 tutorials for multi-robot best practices, plus the "Programming Multiple Robots with ROS 2" book.

Risks & Mitigation strategies

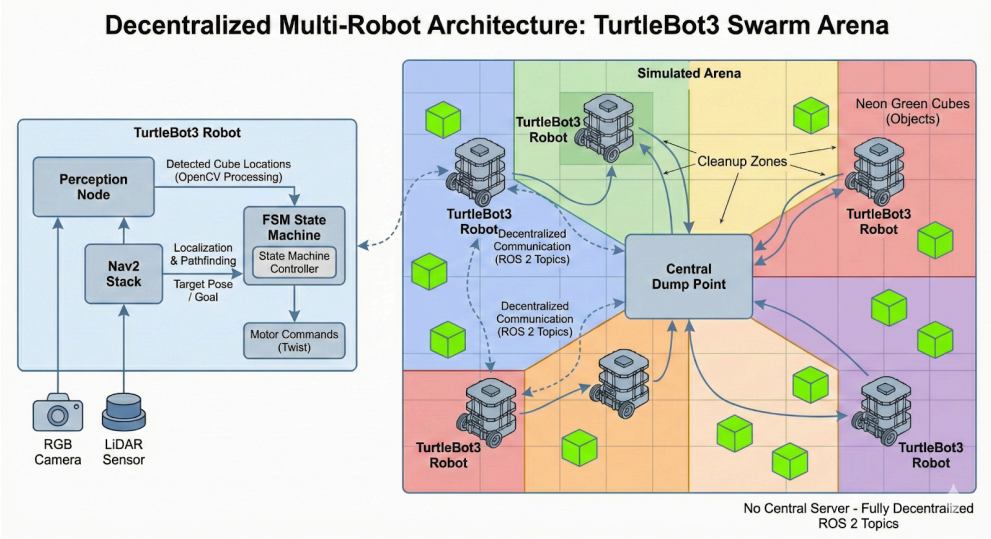
We've identified three main challenges:

1. ***Swarm Deadlocks:*** With 10+ robots, they might crowd each other. Our solution is a "bumping reflex" - if a robot gets blocked by another, it rotates clockwise until the path clears. We'll also design zones to minimize overlap.
2. ***Physics Glitches:*** In simulation, small objects can get stuck under robot wheels. We'll use slightly larger or heavier cubes, tune the physics engine carefully, and test extensively during Week 1.
3. ***Computational Lag:*** Running 10+ cameras and LiDARs at once could slow down the simulation. We'll reduce camera resolution to 320×320, lower the LiDAR update rate (we don't need ultra-high fidelity for color detection), and cap the swarm at 10-12 robots if needed.

Deliverables

1. ***Source code:*** Fully documented ROS 2 package on GitHub covering simulation, perception, navigation, and swarm coordination
2. ***Documentation:*** README with setup instructions, UML diagrams (class, activity, sequence), and Doxygen API docs
3. ***Demo video:*** Full system demo showing robots initializing, searching, and collecting waste across multiple randomized runs
4. ***Test suite:*** Unit and integration tests with coverage documentation
5. ***CI/CD:*** GitHub Actions workflow for automated builds and testing

System Architecture Diagram



References

[1] REMROC - ROS 2 Multi-Robot Coordination. https://github.com/boschresearch/remroc
[2] ROS 2 Navigation Stack. https://navigation.ros.org/
[3] Programming Multiple Robots with ROS 2. https://osrf.github.io/ros2multirobotbook/
[4] REP 105: Coordinate Frames for Mobile Platforms. https://www.ros.org/reps/rep-0105.html
[5] REP 155: Conventions for IMU Sensor Drivers. https://www.ros.org/reps/rep-0155.html