Autonomous Hazardous Waste Foraging System
(Codename: Swarm)

Problem statement: A group of collection robots (>10): Identify and collect objects.
Object locations must not be known to all robots ahead of time and must be randomized each run.

Aakash Shetty Dammala
Group 5

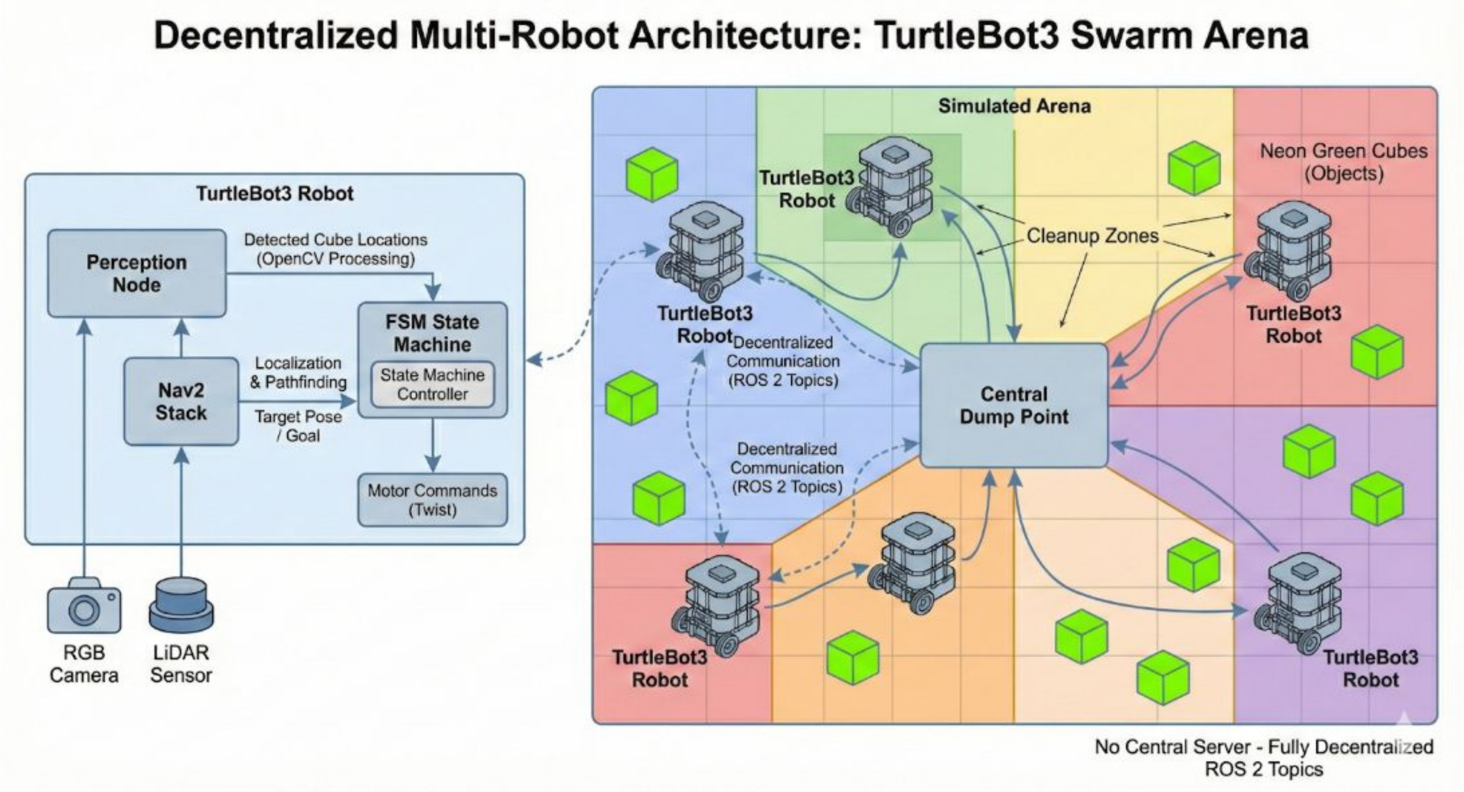
The Problem Statement

- **Scenario**: Disasters like nuclear leaks (e.g., Japan) make manual cleanup dangerous for humans.
- **Business Case**: Opportunity for Acme Robotics to deploy autonomous systems that keep workers out of harm's way.
- **The Challenge**:
  - Robots do not know object locations ahead of time (randomized runs).
  - Perception, Navigation, Grasping

Technical Architecture

- **Hardware & Tools**
  - Robot: E Puck robot.
  - Sensors: 2D LiDAR (obstacle avoidance) and RGB Camera (waste detection).
  - Stack: ROS 2 Humble, Webots Simulation, C++ 14/17.
- **Core Modules**
  - **Perception**: Identifying "hazardous" waste (simulated as neon green objects)
  - **Navigation**: Moving through the environment without collisions.
  - **Grasping**: Hold the object and push it
  - **Collection Strategy**: A Finite State Machine (FSM) to Search - Detect - Grasp - Push - Dump

The Problem Statement



The Design Philosophy

- **Spawn as many robots as possible!**
- The primary goal was to test the limits of the simulation!
- what breaks first?
- and why does it break?

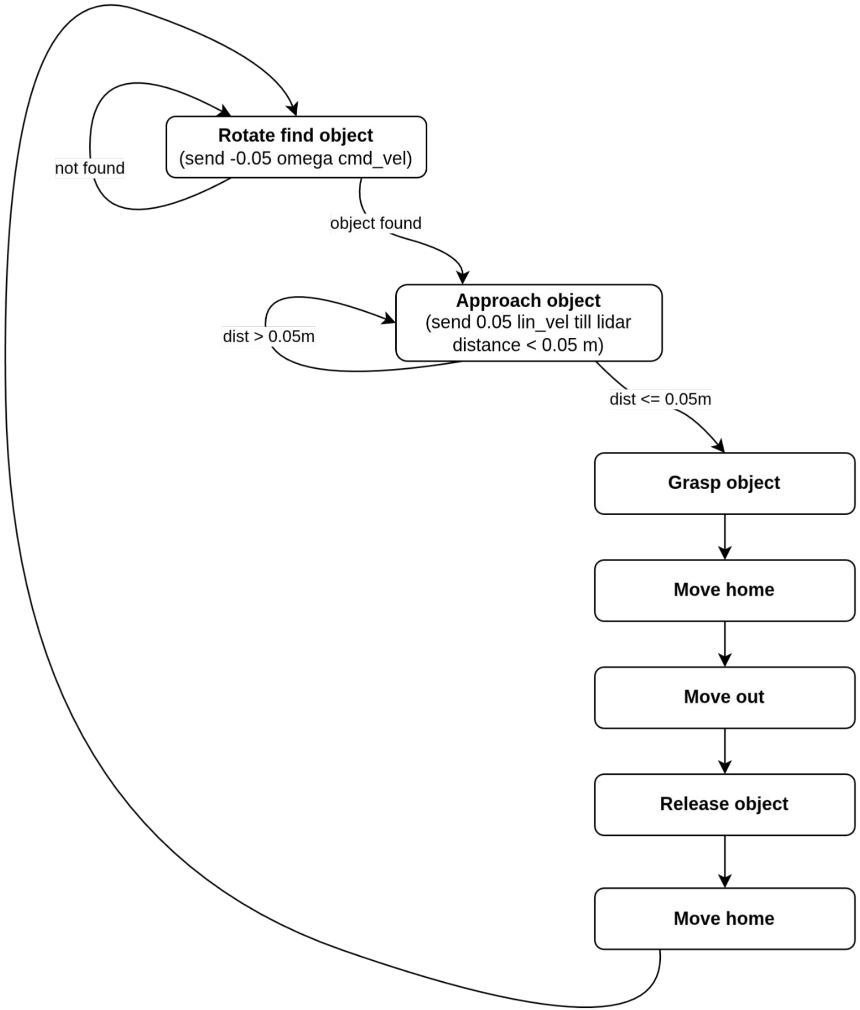
How to achieve the goal of spawning as many robots as possible

- Parametrize Everything:
  - No hardcoded values.
  - swarm_config.yaml drives the simulation (decides robot count).
- Dynamic World Generation: The simulation world is built dynamically at launch based on the config.
- Environment: Each robot is spawned in its own room/zone to prevent complex race conditions and deadlocks.

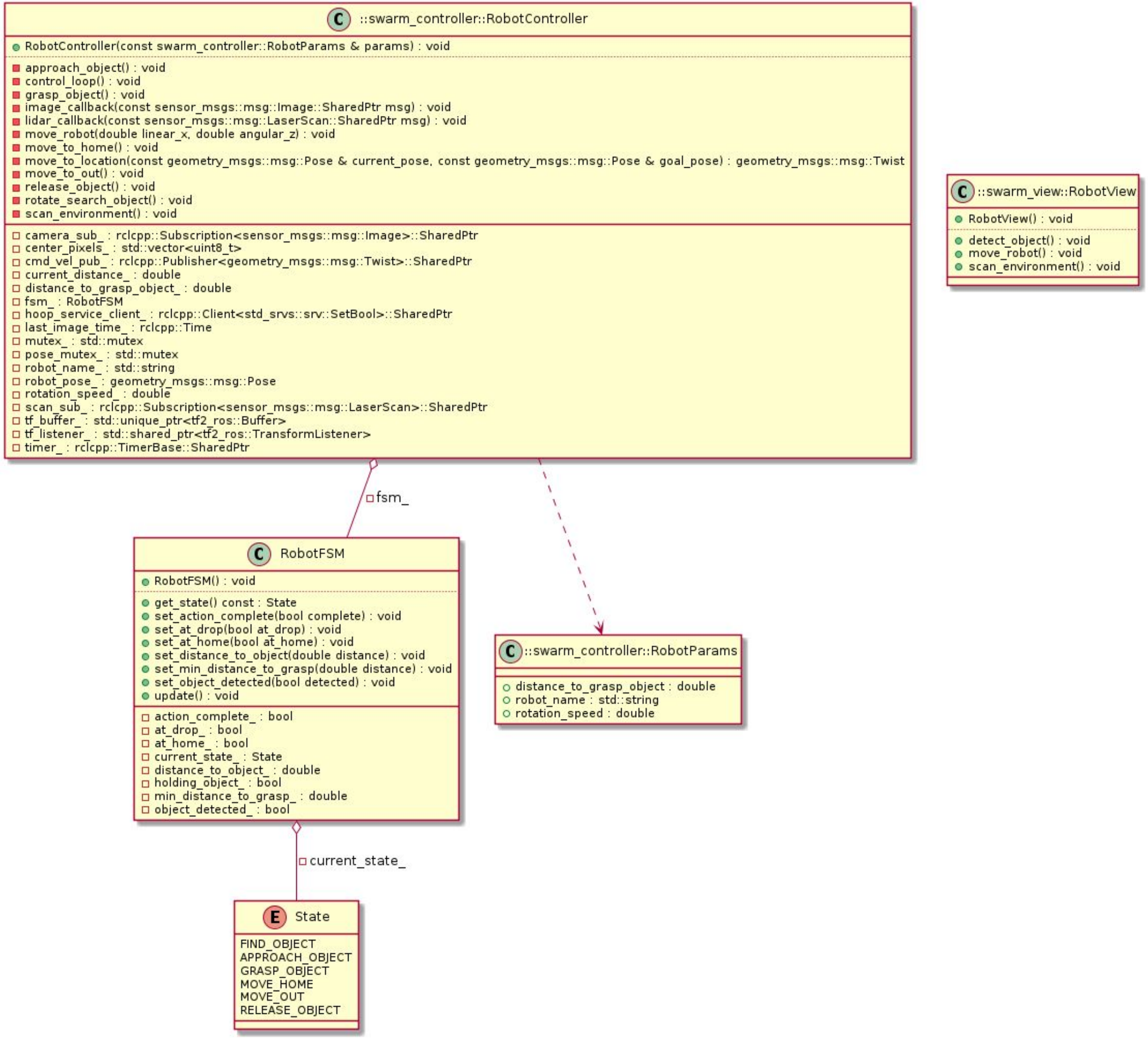
Simplifications made to achieve the goal of spawning as many robots as possible

- Navigation: Switched from the heavy Nav2 stack to a simple custom lightweight planner.
- Switched from OpenCV blob detection to just comparing pixel values.
- Each robot has its own controller node.
- All nodes are spawned in a single executable (to reduce number of auxiliary processes required by a node).

The FSM model



Model View Controller



Simulation demo

1 robot simulation

20 robots simulation

20 robots simulation with gpu

30 robots with gpu

Answering the initial question, what breaks first?

- The bottle next (at least in this case / the way I implemented)
- Bottleneck is ***not Webots simulation, CPU or GPU***
- The ***bottleneck is DDS***, Fast RTPS (default DDS of ROS2 humble)

What could be better

- About 70% of the time was taken by planning and iterating through various iterations of building webots simulation (Building the test bench for the project)
- Only 30% was taken by building the autonomous algorithm
- It was difficult to implement test driven development, especially when I am actively exploring, rapidly building. The picture in my brain was evolving as I was exploring.

The good

- I was able to simulate 30 robots in ROS + Webots
- I was able to answer my initial question of ‘How many robots can I spawn and what breaks first? ’.