Robotics Software Engineer Hiring Assignment: Cooperative Autonomous Warehouse Management (CAWM)

Context: The candidate must implement a **Fleet Management System (FMS)** for a team of heterogeneous Autonomous Mobile Robots (AMRs) operating in a large, dynamic warehouse simulation. The core challenge is maximizing order fulfillment throughput while ensuring collision-free navigation and system safety overrides.

1. Setup and Initialization

- **Robot Models (Heterogeneous Fleet):**
  - **2 x AMR-1 (Tug/Transport):** Fast, non-manipulating robots (like a cart-puller) with a fixed carrying capacity (e.g., 2 units). Equipped with LiDAR and basic IMU.
  - **2 x AMR-2 (Picker/Lift):** Slower, more precise robots (like a forklift/stacker) with a lower carrying capacity (e.g., 1 unit) and a simulated vertical lift or manipulator. Equipped with LiDAR, Depth Camera, and high-precision IMU.
- **Simulation World:** A large, multi-aisle warehouse in Gazebo with:
  - **Storage Racks:** Designated pick-up locations (source).
  - **Packing Stations:** Designated drop-off locations (goal).
  - **Charging Stations:** Designated docking locations.
  - **Dynamic Obstacles:** Randomly spawning/moving "worker" AMRs (simulated traffic) that are *not* part of the candidate's fleet.

2. Core Fleet Management and Task Allocation (Testing Refactoring, Selective Iteration)

1. **Centralized Task Allocation System:**
   - The candidate must implement a central **Order Manager** node that continuously generates a queue of **"Pick-and-Deliver" tasks** (e.g., "Move 2 units from Rack A to Packing Station B").
   - Implement a **Multi-Robot Task Assignment (MRTA)** algorithm (e.g., a simple greedy auction, weighted cost, or Hungarian algorithm) to assign tasks based on:
     - **Robot Heterogeneity:** AMR-1s can only handle tasks of ≤2 units; AMR-2s can only handle ≤1 unit but are required for high-shelf picks (designated Depth Camera-required tasks).
     - **Efficiency:** Minimize the total estimated travel time across the entire fleet (cost function).
2. **Selective Iteration (Dynamic Re-assignment):**
   - **Challenge:** The MRTA algorithm should run a **selective re-assignment iteration** only when a robot's current task completion time is projected to exceed

a threshold (e.g., 150% of the initial estimate) due to congestion or low battery. This directly tests the "selective iteration" experience for system optimization.

---

3. Navigation, Safety, and HAL (Testing Velocity, Override, HAL)

1. **Multi-Agent Path Finding (MAPF) and Collision Avoidance:**
   - Implement a **fleet-aware global planner**. While individual local planners will handle immediate obstacle avoidance, the global path planner must be designed to avoid paths that are currently *predicted* to be high-congestion based on other robots' active global plans.
   - **Traffic Regulation:** Implement a **Traffic Control Node** that enforces one-way movement in narrow aisles and manages intersections to minimize robot-robot waiting time.
2. **Dynamic Motion Smoothing and System Override:**
   - **Robot-Specific Velocity:** Implement a motion smoothing controller where the maximum safe velocity is dynamically scaled based on **both** the robot type (AMR-1 faster than AMR-2) and its current **payload**.
   - **Battery Management Override:** When an AMR's simulated battery level drops below a critical threshold (e.g., 20%), the FMS must **override** its current task assignment, clear its path, and command an immediate, maximum-priority navigation goal to the **nearest Charging Station**.
3. **HAL and Validation Toolkit:**
   - Develop a unified HAL for both robot types that provides a single interface for accessing sensor data.
   - **Validation:** The toolkit must log a warning if the **AMR-2 Depth Camera** reports a distance measurement that is inconsistent with the **AMR-1 LiDAR** measurement when the two robots are in close proximity (simulating a sensor cross-check).

---

4. Code Quality and Deployment

1. **Refactoring for Scalability:** The entire system must be designed to easily scale to **ten or more AMR-1s** by simply changing a single configuration parameter (e.g., an array of robot names in a YAML file or launch argument). The communication and control code must be cleanly **namespaced** and modular.
2. **Yocto/Build Environment:** Provide a **Dockerized ROS/Gazebo environment** that encapsulates all dependencies, simulating the optimized build environment setup for a quick, repeatable deployment, and include a clear, minimal set of environment variables needed to run the simulation.

5. Deployment and Code Quality (Testing Yocto, Refactoring, Codebase)

1. **Environment Setup and Build:**
    - **The candidate must provide a complete, well-documented `README` detailing the steps to launch the entire simulation, navigation, and custom control stack.**
    - **The code must be organized into a proper ROS/ROS 2 workspace using CMake/Colcon and include necessary dependencies for a clean build.**
    - **To submit a screenshare video explaining the various integrities of the problem statement.**

2. **Code Refactoring and Standards:**
    - **All custom nodes developed must adhere to a strict style guide (e.g., Google C++ Style or standard Python PEP 8).**
    - **Deliverable: The candidate must identify one area of the standard ROS Navigation or Gazebo launch files they integrated and propose a refactoring plan (e.g., splitting a monolithic launch file into modular components or converting a complex parameter file into a clean C++ struct/enum representation) and implement a small part of that refactoring.**

Evaluation Criteria

Skill Area Tested	Relevant Experience Point	Success Metrics
SLAM & Global Planning	***Mapping its own environment***	**Generates a complete, high-quality map; successfully navigates from base to rescue zone.**
Safety & Override	***Obstacle detection, system override***	**Robot correctly performs a controlled stop or evasive maneuver when an obstacle violates**

		the safe distance constraint, overriding user input, Dynamic obstacles.
Motion Control	***Motion smoothing based on dynamic velocity***	**Robot's movement is observably smooth; velocity profile correctly modulates based on environment type.**
Custom Integration	***Developed HAL and validation toolkit***	**A functional HAL class is used by the navigation node; the validation checks correctly run and log outputs for the IMU and Camera.**
System Optimization	***Improved global mapping... selective iteration***	**Demonstrated ability to programmatically restrict the map update area based on a defined condition.**
Code Quality & Build	***Refactoring and minor functional updates***	**Clean, well-commented, and modular code; workspace builds cleanly; provides a compelling refactoring demonstration.**