Multi-Robot Collaboration Simulation Plan

Strategic Overview

Goal: Create a clean, publishable multi-robot collaboration simulation that demonstrates distributed coordination, task allocation, and performance under various conditions - independent of any specific system.

Philosophy: Build a credible testbed that robotics researchers would accept as valid evaluation infrastructure, even without physical robots or ROS2.

Phase 1: Scenario Design & Validation

Milestone 1.1: Select Core Collaboration Scenario

Three Strong Options:

Option A: Multi-Robot Coverage/Exploration

- **Task**: N robots must explore/inspect unknown 2D environment
- **Collaboration aspects**: Area division, frontier coordination, map merging
- **Metrics**: Time to full coverage, duplicate work percentage, communication overhead
- **Complexity**: Low-Medium
- **Research relevance**: Search & rescue, environmental monitoring

Option B: Cooperative Object Transport

- **Task**: Multiple robots must coordinate to move large/heavy objects
- **Collaboration aspects**: Formation control, load balancing, synchronization
- **Metrics**: Transport time, energy consumption, coordination failures
- **Complexity**: Medium
- **Research relevance**: Warehouse automation, construction

Option C: Distributed Task Allocation (Heterogeneous Fleet)

- **Task**: Different robot types execute complex multi-stage jobs
- **Collaboration aspects**: Capability matching, resource sharing, dynamic reallocation
- **Metrics**: Makespan, robot utilization, task failure rate
- **Complexity**: Medium-High

- **Research relevance**: Smart manufacturing, logistics

Recommendation: Choose **Option C -** most generalizable and demonstrates widest range of coordination challenges.

Milestone 1.2: Define Detailed Scenario Specification

Scenario: Warehouse Inspection & Maintenance

Environment:

- 50m × 30m warehouse grid (1m resolution)
- 20-30 inspection zones (shelves, machinery, loading docks)
- 3-5 maintenance stations (for repairs, battery charging)
- Static obstacles (walls, equipment)

Robot Fleet (12-32 robots):

- **Type I: Inspectors** (40% of fleet)
  - Equipped with cameras, lightweight
  - Speed: 1.5 m/s
  - Capability: Visual inspection, QR scanning
  - Battery: 60 minutes operational
- **Type II: Repair Units** (30% of fleet)
  - Equipped with tools, medium weight
  - Speed: 1.0 m/s
  - Capability: Minor repairs, component replacement
  - Battery: 90 minutes operational
- **Type III: Data Collectors** (30% of fleet)
  - Equipped with sensors (thermal, acoustic)
  - Speed: 0.8 m/s
  - Capability: Detailed diagnostics, data logging
  - Battery: 120 minutes operational

Task Types:

1. **Routine Inspection**: Inspect zone → Report status (5-10 min)
2. **Anomaly Investigation**: Inspect → Diagnose → Report (10-20 min, may require Type III)
3. **Preventive Maintenance**: Inspect → Minor repair (15-30 min, requires Type II)
4. **Emergency Response**: Detect issue → Investigate → Repair → Verify (20-40 min, multi-robot)

Task Properties:

- Dependencies: Some tasks prerequisite for others

- Urgency levels: Routine (low), Scheduled (medium), Emergency (high)
- Resource requirements: Battery, capabilities, possibly multiple robots

Milestone 1.3: Establish Baseline Coordination Strategies

Strategy 1: Centralized Coordinator

- Single master robot assigns all tasks
- Global knowledge of robot states
- Optimal allocation (greedy or auction-based)

Strategy 2: Fully Distributed (Market-based)

- Robots bid on tasks independently
- No central authority
- Convergence through negotiation

Strategy 3: Hierarchical (Your proposed approach)

- Robots organized in dynamic groups
- Group leaders coordinate locally
- Inter-group coordination for complex tasks

Strategy 4: Reactive (Baseline)

- Nearest available robot takes next task
- No lookahead or optimization
- Minimal coordination

Phase 2: Mathematical Foundations

Milestone 2.1: Formalize Task Model

Task Representation:

- Task ID, type, location
- Required capabilities: `C_req ⊆ {vision, manipulation, sensing, ...}`
- Estimated duration: `d_est` (with uncertainty)
- Deadline: `t_deadline` (soft or hard)
- Dependencies: Directed acyclic graph (DAG) of prerequisite tasks
- Priority/utility: Numerical value

Task Arrival Process:

- Poisson process with rate λ (tasks/minute)
- Mix: 60% routine, 25% investigation, 10% maintenance, 5% emergency
- Temporal patterns: Morning rush (λ=8), midday (λ=4), evening (λ=6)

Milestone 2.2: Formalize Robot Model

Robot State Vector:

- Position: `(x, y)`
- Battery level: `b ∈ [0, 100]%`
- Capabilities: `C_robot ⊆ {vision, manipulation, ...}`
- Current task: `τ_current` or idle
- Task queue: Ordered list of assigned future tasks
- Communication range: `r_comm` (e.g., 20m for local, ∞ for WiFi)

Robot Dynamics:

- Simple kinematic model: Straight-line movement at constant speed
- Battery drain: `db/dt = -k_move * v - k_idle` (moving vs. idle)
- Task execution: Probabilistic completion time `d_actual ~ N(d_est, σ)`

Milestone 2.3: Define Coordination Metrics

Primary Metrics:

1. **Makespan**: Time to complete all tasks
2. **Average Task Completion Time**: Mean time from task arrival to completion
3. **Robot Utilization**: `(active_time) / (total_time)` per robot
4. **Travel Distance**: Total distance traveled by all robots
5. **Energy Consumption**: Total battery drain

Secondary Metrics: 6. **Communication Overhead**: Number of messages exchanged 7. **Task Deadline Violations**: Percentage of tasks missing deadlines 8. **Load Imbalance**: Variance in utilization across robots 9. **Idle Time**: Time robots spend waiting for tasks 10. **Coordination Latency**: Time to assign task after arrival

Failure Metrics: 11. **Task Failure Rate**: Tasks abandoned due to battery/capability issues 12. **Resilience**: Performance degradation when robots fail

---

Phase 3: Core Simulation Engine

Milestone 3.1: Environment Implementation

Grid World:

- Numpy 2D array for occupancy (0=free, 1=obstacle)
- Spatial indexing for fast proximity queries
- Visualization: Matplotlib with live updates (optional animation)

Key Methods:

- `is_path_free(start, end)`: Simple line-of-sight check
- `get_distance(pos1, pos2)`: Euclidean or Manhattan distance
- `get_nearest_station(robot_pos, station_type)`: For charging/maintenance
- `place_tasks(num_tasks, distribution)`: Generate task locations

Milestone 3.2: Robot Agent Implementation

Agent Class Structure:

- **Attributes**: State vector (position, battery, capabilities, etc.)
- **Behaviors**:
    - `move_to(target_pos, dt)`: Update position each time step
    - `execute_task(task, dt)`: Simulate task execution with progress tracking
    - `update_battery(dt)`: Drain battery based on activity
    - `can_execute(task)`: Check capability and battery feasibility
    - `estimate_completion_time(task)`: Predict when task would finish

Decision-Making:

- Input: Current state, available tasks, messages from other robots
- Output: Next action (move, execute, wait, communicate)
- Implementation: Depends on coordination strategy (will vary per baseline)

Milestone 3.3: Task Generator

Workload Profiles:

- **Light Load**: λ=3 tasks/min, 10-15 robots
- **Medium Load**: λ=6 tasks/min, 15-20 robots
- **Heavy Load**: λ=10 tasks/min, 20-30 robots
- **Burst Load**: Alternate λ=2 and λ=15 every 5 minutes

Task Generation:

- Draw inter-arrival times from exponential distribution
- Sample task type from defined mix (60% routine, etc.)
- Assign location randomly or clustered (zones have different demand)
- Generate dependencies: 20% of tasks have prerequisite tasks

Milestone 3.4: Discrete Event Simulation Loop

Time-Stepped Simulation:

- Time step: Δt = 0.1 seconds (balance accuracy vs. speed)
- Each step:
    1. Generate new tasks (if arrival time reached)
    2. Update robot positions/battery
    3. Progress task execution
    4. Execute coordination algorithm (assign new tasks)
    5. Handle task completions
    6. Log metrics
    7. Check termination condition

Event Queue (alternative to time-stepping):

- Events: Task arrival, task completion, robot reaches destination, battery depleted
- Process events in chronological order
- More efficient for sparse scenarios

Termination:

- Fixed duration (e.g., 2 hours simulated time)
- All tasks completed
- Performance threshold reached

Phase 4: Coordination Algorithms (Days 11-15)

Milestone 4.1: Centralized Coordinator

Algorithm:

- Maintain global task queue sorted by priority/deadline
- When robot becomes idle:
    1. Iterate through unassigned tasks
    2. For each task, compute cost for each idle robot: `cost = travel_time + execution_time`

3. Assign task to robot minimizing cost
4. Update robot's task queue

Optimization:

- Use Hungarian algorithm for optimal bipartite matching (robots ↔ tasks)
- Preemption: Allow reassignment if high-priority task arrives

Metrics to Track:

- Assignment computation time (should increase with fleet size)
- Optimality gap vs. brute force (for small instances)

Milestone 4.2: Market-Based Coordination

Algorithm (Contract Net Protocol variant):

- When task arrives:
    1. Broadcast task announcement to all robots in range
    2. Robots compute bid: `bid = 1 / (travel_time + execution_time + queue_delay)`
    3. Highest bidder wins task
    4. Handle conflicts (multiple tasks awarded simultaneously)

Enhancements:

- Multi-round bidding for complex tasks
- Bid includes capability matching score
- Auction timeout to prevent deadlock

Metrics to Track:

- Number of messages per task assignment
- Convergence time (auction duration)
- Solution quality vs. centralized

Milestone 4.3: Hierarchical Coordination

Algorithm:

- Organize robots into K clusters (spatial or capability-based)
- Each cluster has elected leader (e.g., highest battery robot)
- Two-level assignment:
    1. **Inter-cluster**: Tasks assigned to clusters based on proximity/capability
    2. **Intra-cluster**: Cluster leader assigns to specific robot

Cluster Formation:

- K-means on robot positions (spatial clustering)
- Or group by capability (all Type I together)
- Dynamic re-clustering every T minutes

Leader Election:

- Highest battery level among cluster members
- Re-elect if leader battery <20% or leader fails

Metrics to Track:

- Cluster stability (re-clustering frequency)
- Leader communication overhead
- Performance vs. fully centralized

Milestone 4.4: Reactive Baseline

Algorithm:

- Each robot independently scans for nearest unassigned task
- Claim task if capable and battery sufficient
- No coordination, possible conflicts resolved randomly

Purpose: Worst-case baseline to show value of coordination

Phase 5: Advanced Features (Days 16-18)

Milestone 5.1: Dynamic Task Migration

Scenario: Robot assigned task but better option appears

Migration Triggers:

- Higher priority task arrives near robot's current location
- Robot's battery becomes critically low mid-task
- Robot fails or becomes unavailable

Migration Protocol:

- Estimate cost of migration: `transfer_overhead + new_assignment_benefit`
- Migrate only if net benefit >threshold

- Update task state (preserve partial progress if applicable)

Metrics:

- Migration frequency
- Performance improvement from migration
- Overhead (task delays due to migration)

Milestone 5.2: Multi-Robot Task Execution

Scenario: Tasks requiring 2+ robots simultaneously

Coordination Challenges:

- Rendezvous: Robots must arrive at task location within time window
- Synchronization: Task starts only when all required robots present
- Load balancing: Avoid assigning all capable robots to single task

Implementation:

- Task specifies: `num_robots_required`, `capability_requirements[]`
- Coordinator assigns robot team
- Robots negotiate arrival time, wait for stragglers
- Task execution time split among participants

Metrics:

- Waiting time (time robots spend idle at rendezvous)
- Task completion speedup (vs. single robot)

Milestone 5.3: Battery Management & Recharging

Battery Model:

- Drain rate: Moving (high), executing task (medium), idle (low)
- Recharge: Robot travels to charging station, charges at fixed rate (e.g., 20%/min)

Recharge Policies:

- **Reactive**: Recharge when battery <10%
- **Proactive**: Recharge when battery <30% and no urgent tasks
- **Opportunistic**: Recharge during idle periods if station nearby

Trade-offs:

- Too conservative → wasted capacity, frequent recharging

- Too aggressive → risk of depletion mid-task

Metrics:

- Number of recharge trips
- Time spent recharging
- Emergency depleted batteries (failures)

Milestone 5.4: Communication Constraints

Limited Range:

- Robots can only communicate within `r_comm` (e.g., 20m)
- Requires multi-hop routing for distant robots
- Partition tolerance: Separate clusters can't coordinate

Message Delays:

- Wireless latency: 10-100ms per message
- Bandwidth limits: Maximum N messages per second
- Packet loss: 5% probability

Impact on Coordination:

- Market-based: Slower convergence, incomplete bids
- Centralized: Controller may have stale state
- Hierarchical: Inter-cluster coordination delayed

Metrics:

- Message delivery rate
- Coordination latency increase
- Performance degradation vs. ideal communication

Phase 6: Experimental Design (Days 19-21)

Milestone 6.1: Baseline Comparisons

Experiment 1: Scalability

- **Variable**: Number of robots (4, 8, 12, 16, 24, 32)
- **Fixed**: Task arrival rate (λ=6), duration (2 hours)
- **Measure**: Makespan, average completion time, utilization

- **Expectation**: Centralized degrades at high N, distributed scales better

Experiment 2: Load Sensitivity

- **Variable**: Task arrival rate (λ=2, 4, 6, 8, 10, 12)
- **Fixed**: 16 robots, 2-hour duration
- **Measure**: Task queue length, deadline violations, robot idle time
- **Expectation**: All strategies degrade beyond saturation point, but at different rates

Experiment 3: Heterogeneity Impact

- **Variable**: Fleet composition (homogeneous vs. mixed types)
- **Fixed**: 16 robots, λ=6
- **Measure**: Task failure rate, capability utilization
- **Expectation**: Heterogeneous fleet benefits from intelligent matching

Milestone 6.2: Robustness Testing

Experiment 4: Robot Failures

- **Failure modes**:
  - Random: Each robot fails with 1% probability per minute, recovers after 5 minutes
  - Correlated: 20% of fleet fails simultaneously at t=30min
- **Measure**: Task reassignment time, completion time increase, failure propagation

Experiment 5: Communication Disruptions

- **Scenarios**:
  - Partitioned network: Fleet splits into 2 disconnected groups
  - High latency: 500ms message delay
  - Packet loss: 20% messages dropped
- **Measure**: Coordination effectiveness, duplicate work, convergence time

Experiment 6: Dynamic Task Patterns

- **Patterns**:
  - Spatial hotspots: 80% tasks arrive in 20% of warehouse area
  - Temporal bursts: Sudden spike in emergency tasks
- **Measure**: Load balancing quality, response time for high-priority tasks

Milestone 6.3: Ablation Studies

Ablation 1: Task Migration

- Compare with vs. without migration for hierarchical strategy

- **Isolate**: Benefit of dynamic reassignment

Ablation 2: Battery Management

- Compare reactive vs. proactive vs. opportunistic recharging
- **Isolate**: Impact of intelligent battery planning

Ablation 3: Communication Range

- Vary `r_comm`: 10m, 20m, 50m, ∞ (WiFi)
- **Isolate**: Communication requirements for coordination

Milestone 6.4: Parameter Sensitivity

Key Parameters to Vary:

- Task execution uncertainty: σ=0 (deterministic), σ=0.2*mean, σ=0.5*mean
- Robot speed: ±20% variation
- Battery capacity: 40min, 60min, 90min, 120min
- Task complexity distribution: More vs. fewer multi-robot tasks

Objective: Show results are robust, not tuned to specific settings

---

Phase 7: Metrics & Analysis (Days 22-24)

Milestone 7.1: Performance Metrics Collection

Efficiency Metrics:

- **System Throughput**: Tasks completed per hour
- **Makespan**: Total time to complete all tasks
- **Average Turnaround Time**: Mean (completion_time - arrival_time)
- **Robot Utilization**: Active time / (active + idle + recharge)

Quality Metrics:

- **Deadline Violation Rate**: % tasks completed late
- **Task Failure Rate**: % tasks abandoned (battery/capability issues)
- **Opportunity Cost**: Tasks not started due to resource unavailability

Coordination Metrics:

- **Assignment Latency**: Time from task arrival to robot assignment

- **Communication Overhead**: Total messages sent
- **Convergence Time**: Time for distributed algorithms to reach consensus

Fairness Metrics:

- **Load Imbalance**: Gini coefficient of task distribution
- **Energy Imbalance**: Variance in battery levels at end
- **Distance Imbalance**: Variance in travel distances

Milestone 7.2: Statistical Analysis

Methodology:

- Run each configuration 30 times with different random seeds
- Report mean ± 95% confidence interval
- Statistical tests:
    - T-tests for pairwise comparisons (centralized vs. hierarchical)
    - ANOVA for multi-group comparisons (4 strategies)
    - Effect size (Cohen's d) to quantify practical significance

Handling Outliers:

- Identify outliers ($>3\sigma$ from mean)
- Investigate causes (bugs vs. legitimate edge cases)
- Report with/without outliers if difference >10%

Milestone 7.3: Key Visualizations

Figure 1: Scalability Comparison

- Line plot: X=number of robots, Y=makespan
- 4 lines (one per strategy), error bars for confidence intervals
- **Insight**: Show which strategy scales best

Figure 2: Load Sensitivity

- Line plot: X=task arrival rate, Y=average completion time
- Annotate saturation point (where performance degrades rapidly)
- **Insight**: Operating limits for each strategy

Figure 3: Communication Overhead

- Bar chart: X=strategy, Y=messages per task
- Grouped by robot count (8, 16, 24)
- **Insight**: Cost of coordination

Figure 4: Robustness to Failures

- Box plot: X=failure scenario, Y=completion time increase
- Compare strategies side-by-side
- **Insight**: Which strategy is most resilient

Figure 5: Utilization Heatmap

- 2D heatmap: Rows=robots, Columns=time bins, Color=utilization %
- Compare centralized (even) vs. reactive (uneven)
- **Insight**: Load balancing quality

Figure 6: Task Completion CDF

- Cumulative distribution: X=completion time, Y=% tasks completed
- Overlay strategies
- **Insight**: Tail latency (P99, P95)

Phase 8: Validation & Verification (Days 25-26)

Milestone 8.1: Sanity Checks

Conservation Laws:

- Number of tasks generated = completed + in_progress + failed
- Total battery consumed ≤ initial battery × num_robots + recharged_battery
- Total distance traveled ≥ straight-line distance to all task locations

Boundary Conditions:

- Single robot: Should match optimal single-agent policy
- Infinite robots: Makespan should equal longest task duration
- No tasks: Robots should remain idle, zero overhead

Regression Tests:

- Small known scenarios with hand-calculated optimal solutions
- Verify simulator matches expected outcome

Milestone 8.2: Realism Assessment

Compare with Theoretical Bounds:

- Lower bound makespan: `max(sum(task_durations) / num_robots, longest_task)`

- Upper bound: Naive sequential execution time
- Your results should fall between these

Literature Comparison:

- Find similar simulations in recent papers (ICRA, IROS, etc.)
- Compare your task completion times, utilization rates
- Should be within 20-30% (different assumptions, but order of magnitude similar)

Qualitative Validation:

- Show simulation videos to robotics experts
- Ask: "Does robot behavior look reasonable?"
- Common issues: Unnatural trajectories, excessive waiting, poor task distribution

Milestone 8.3: Sensitivity to Implementation Details

Discretization Error:

- Compare Δt=0.05s vs. 0.1s vs. 0.5s
- Results should vary <5% (if larger, need finer time steps)

Random Number Generation:

- Test with different RNG seeds (already covered in 30 runs)
- Verify no seed-dependent artifacts

Numerical Precision:

- Check for floating-point issues (e.g., battery never exactly 0.0)
- Use epsilon comparisons where appropriate

Phase 9: Documentation & Reproducibility (Days 27-28)

Milestone 9.1: Code Documentation

README.md:

- Project overview and goals
- Installation instructions (Python version, dependencies)
- Quick start example
- How to reproduce each figure in results

API Documentation:

- Docstrings for all classes and key methods
- Type hints for function signatures
- Usage examples in docstrings

Configuration Files:

- YAML/JSON for all tunable parameters
- Separate configs for each experimental scenario
- Commented explanations of parameter meanings

Phase 10: Advanced Extensions (Optional, Days 29-30)

Milestone 10.1: Learning-Based Coordination

Approach: Train coordination policy with reinforcement learning

Setup:

- State: Robot states, task queue, environment map
- Actions: Task assignments or movement decisions
- Reward: Negative makespan or completion time
- Algorithm: Multi-agent RL (e.g., QMIX, MAPPO)

Comparison: RL-learned policy vs. hand-crafted heuristics

Challenge: Requires significant compute, may need simplified environment

Milestone 10.2: 3D Visualization

Tools: Pygame or PyBullet (without physics, just rendering)

Features:

- Top-down view of warehouse
- Robots as colored dots/icons
- Task locations as markers
- Battery level indicators
- Task assignment animations (lines connecting robots to tasks)

Purpose: Compelling demos for presentations

Milestone 10.3: Real-World Trace Integration

Data Sources:

- Publicly available warehouse robot logs (Amazon, Fetch Robotics datasets if accessible)
- Extract task arrival patterns, locations, durations
- Use as realistic workload generator

Validation: Does your simulation produce similar metrics to real systems?

Deliverables Timeline

Week 1 (Days 1-7):

✅ Scenario fully defined
✅ Mathematical models formalized
✅ Core simulation engine implemented
✅ Basic visualization working

Week 2 (Days 8-14):

✅ All 4 coordination strategies implemented
✅ Battery management and multi-robot tasks functional
✅ Initial experiments running

Week 3 (Days 15-21):

✅ All experiments completed (30 runs each)
✅ Statistical analysis done
✅ All figures generated

Week 4 (Days 22-28):

✅ Validation complete
✅ Code documented and packaged
✅ Reproducibility verified
✅ Results written up
