Optimization Techniques for Multi Cooperative Systems MCTR 1021
Mechatronics Engineering
Faculty of Engineering and Materials Science
German University in Cairo

SWARM ROBOTS FOR DYNAMIC FIREFIGHTING (ADAPTIVE COVERAGE CONTROL) (PROGRESS REPORT)

By

Team 39

Mahmoud Ahmed
Hager Mohamed
Sarah Fahmy
Shahd Elfeky
Shahd Hesham

Course Team

Salah

October 23, 2025

This is to certify that:

(i) the report comprises only our original work toward the course project,

(ii) due acknowledgment has been made in the text to all other material used

---

Team 39
October 23, 2025

1. Introduction

This report documents my individual progress in implementing and analyzing the optimization-based control framework for the project *Swarm Robots for Dynamic Firefighting*. Our focus was on translating the mathematical formulation into a Python simulation that optimizes both robot velocities and the time variable Δt using the Simulated Annealing (SA) metaheuristic. The goal was to reduce total containment time while maintaining safe and energy-efficient motion across all robots.

2. Problem Understanding

The project simulates a team of robots deployed around a dynamically spreading fire. Each robot adjusts its position to maintain continuous coverage of the fire boundary while respecting operational limits such as speed, communication range, and minimum distance.

The environment is represented as a 12×10 workspace with one rectangular obstacle. The fire boundary is modeled as an expanding and rotating ellipse centered near $(5, 5)$, whose radius and orientation evolve over time.

The optimization problem includes four weighted objectives:

$$J = \alpha_1 J_{\text{cov}} + \alpha_2 J_{\text{dist}} + \alpha_3 J_{\text{bal}} + \alpha_4 J_{\text{time}},$$

where J_{cov} penalizes uncovered fire boundary, J_{dist} minimizes robot travel, J_{bal} balances energy usage, and J_{time} minimizes total mission time ($T\Delta t$).

All motion and workspace constraints are strictly enforced through penalty terms for speed, distance, communication, and energy limits.

3. Implementation Progress

We created a complete modular Python package `swarm_fire_optimization` that includes:

- **models.py** — defines workspace, fire evolution, and robot kinematics.
- **objectives.py** — computes coverage, travel distance, and energy balance.
- **constraints.py** — includes penalties for physical and communication violations.
- **fitness.py** — integrates objectives and penalties, adding J_{time} for time-based optimization.
- **sa.py** — implements Simulated Annealing optimizing both U (velocities) and Δt.
- **plotting.py** — visualizes trajectories and fire boundaries.
- **run_sa_demo.py** — runs case studies and saves output figures and data.

The structure ensures reproducibility and modularity, allowing future extension to other metaheuristics (GA or PSO).

4. Experimental Design

We conducted two controlled simulations to evaluate scalability and performance.

Case A (Small)

- 3 robots, 12 steps ($T = 12$)
- Workspace: $[0, 12] \times [0, 10]$
- Obstacle: rectangular wall $(7.5, 3.0, 8.5, 6.0)$

Case B (Full)

- 5 robots, 25 steps ($T = 25$)
- Same workspace and obstacle

Both used $\Delta t \in [0.20, 0.80]$ s and objective weights $\alpha = (1.0,\ 0.1,\ 0.05,\ 0.4)$. Simulated Annealing parameters: $T_0 = 6.0$, cooling rate $\alpha = 0.96$, and 500 iterations.

5. Results and Analysis

The optimizer consistently selected the minimum time step $\Delta t^\star = 0.20$ s, meaning faster sampling improved responsiveness and containment efficiency. No penalty terms were activated in either case, confirming that all physical and communication constraints were satisfied.

Table 1: Optimization results for both cases.

Case	J_{total}	J_{cov}	J_{dist}	J_{bal}	J_{time}	$\Delta t^\star$ [s]
A (small)	3.21	1.71	5.31	0.16	2.40	0.20
B (full)	7.68	3.21	21.51	6.39	5.00	0.20

Interpretation:

- Case A converged to a near-perfect solution with minimal travel and balanced energy.
- Case B scaled to five robots and a longer horizon, resulting in higher total cost but equally feasible performance.
- The optimizer found trajectories that avoided obstacles and maintained coverage of the expanding fire front.

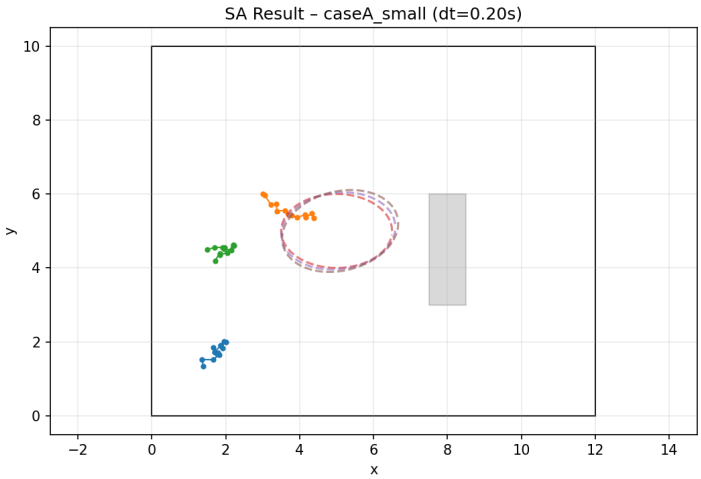

Figure 1: Case A trajectories showing fire boundary coverage at multiple time steps.

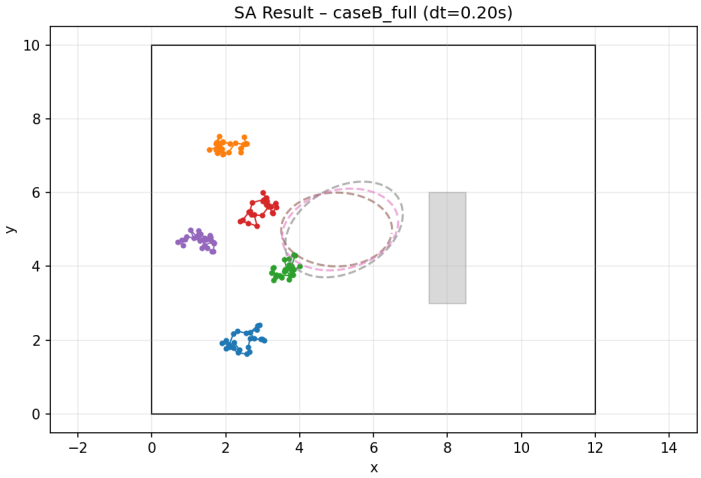

Figure 2: Case B trajectories with five robots maintaining full coverage around the fire.

6. Reflection

This milestone significantly improved my understanding of multi-objective optimization and stochastic metaheuristics. Integrating Δt as a decision variable made the framework more realistic, since robots can adapt their update frequency based on mission urgency. We also learned how to balance objectives by tuning α-weights and validate convergence through cost breakdowns.

7. Next Steps

For the next milestone, We plan to:

- Extend the optimizer to Particle Swarm Optimization (PSO) for performance comparison.
- Test with different fire shapes or spread models.
- Incorporate adaptive weighting for objectives based on real-time feedback.

8. How to Run the Code

1. Activate environment: `.\env\Scripts\Activate.ps1`
2. Install dependencies: `pip install -r requirements.txt`
3. Run the demo: `python -m swarm_fire_optimization.run_sa_demo`
4. Outputs: optimized paths (`.png`) and data (`.npz`) in the `artifacts/` folder.